sun® microsystems

System Administration *for the* Sun Workstation®

Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: 800-1323-03
Revision B of 17 February 1986

Credits and Trademarks

Sun Workstation® is a registered trademark of Sun Microsystems, Inc.

SunStation®, Sun Microsystems®, SunCore®, SunWindows®, DVMA®, and the combination of Sun with a numeric suffix are trademarks of Sun Microsystems, Inc.

UNIX, UNIX/32V, UNIX System III, and UNIX System V are trademarks of AT&T Bell Laboratories.

Intel® and Multibus® are registered trademarks of Intel Corporation.

DEC®, PDP®, VT®, and VAX® are registered trademarks of Digital Equipment Corporation.

The appendices in the final section of this manual were originally derived from the following works:

Fsck — The UNIX File System Check Program
was originally written by T. J. Kowalski, Bell Laboratories, Murray Hill, New Jersey; and was revised by Marshall Kirk McKusick, Computer Systems Research Group, University of California at Berkeley.

Sendmail — An Internetwork Mail Router
was originally written by Eric Allman, formerly of Project Ingres, University of California at Berkeley.

Sendmail — Installation and Operation Guide
was originally written by Eric Allman, formerly of Project Ingres, University of California at Berkeley.

Uucp Implementation Description
was originally written by D. A. Nowitz, Bell Laboratories, Murray Hill, New Jersey.

Copyright © 1986 by Sun Microsystems.

This publication is protected by Federal Copyright Law, with all rights reserved. No part of this publication may be reproduced, stored in a retrieval system, translated, transcribed, or transmitted, in any form, or by any means manual, electric, electronic, electro-magnetic, mechanical, chemical, optical, or otherwise, without prior explicit written permission from Sun Microsystems.

Contents

Chapter 1 Introduction And General Advice 3

Chapter 2 Sun Network Services 7

2.1. Introduction And Terminology 7

Networking Models 7

Terminology 8

UNIX Meets Sun Network Services 8

A Hint About Debugging UNIX In The Network Environment 9

2.2. nd: The Sun Network Disk Service 10

Partitions 10

The `/etc/nd.local` File 10

The Server's View Of Partitions 13

The Client's View Of Partitions 13

Using Clients' Root Partitions from the Server 13

Using Clients' Swap Partitions From The Server 16

More Advanced Use 16

Words of Warning 17

2.3. NFS: The Network File System Service 17

What Is The NFS Service 17

How The NFS Works 18

How To Become An NFS Server 18

How To Remote Mount A File System 19

Typical NFS Layout 19

Debugging the Network File System 22

General Hints 22
Remote Mount Failed 24
Programs Hung 28
Hangs Part Way Through Boot 29
Everything Works But Slowly 29
Incompatibilities with Earlier Versions 30
No Superuser Access Over The Network 30
File Operations Not Supported 33
Cannot Access Remote Devices 33
Beefing Up NFS Security 33
Port Monitoring 33
IP Source Address Checking 33
Clock Skew In User Programs 34
2.4. yp: The Sun Yellow Pages Service 35
What Is The Yellow Pages Service? 35
The yp Map 36
The yp Domain 36
Masters And Slaves 36
Yellow Pages Overview 36
Yellow Pages Installation and Administration 38
How To Set Up A Master yp Server 38
Altering A yp Client's Files To Use yp Services 39
How To Set Up A Slave yp Server 40
How To Set Up A yp Client 41
How To Modify Existing yp Maps After yp Installation 41
Propagation Of A yp Map 43
How To Make New yp Maps After yp Installation 44
How To Add A New yp Server Not In The Original Set 45
How To Change The Master Server 45
Debugging A Yellow Pages Client 46
On Client: Commands Hang 46
On Client: yp Service Unavailable 48
On Client: ypbind Crashes 48

On Client: `ypwhich` Inconsistent 49
Debugging A Yellow Pages Server 49
On Server: Different Versions Of A `yp` Map 50
On Server: `ypserv` Crashes 51
Yellow Pages Policies 52
How Security Is Changed With The Yellow Pages 52
Global And Local `yp` Database Files 52
Two Other Files `yp` Consults 53
Security Implications 53
Special `yp` Password Change 54
Netgroups: Networkwide Groups Of Machines And Users 54
Manual Pages Covering Security Issues 55
What If You Do Not Use The Yellow Pages? 56
2.5. Adding A New User To A Machine 56
Edit The Master `yp` Server's `/etc/passwd` File 56
Make A Home Directory 58
The New User's Environment 58
2.6. Adding A Client Machine To An `nd` Server 59
Choose A Unique Client Name 59
Edit Two Files On The Master `yp` Server 59
Edit The `nd` Server's `/etc/nd.local` File 60
Preparing An Unused Partition For A New Client 62
Preparing A Previously Used Client Partition 64
Booting The New Client Machine 65

Chapter 3 Disks And File Systems **69**
3.1. Disk Device Terminology 69
Controllers, Device Drivers, Units 70
SCSI Disks vs SMD Disks 70
SMD Disk Formatting, Mapping, Slipping 71
SCSI/ST-506 Disk Formatting, Mapping, Slipping 72
Labels And Partitions 73
3.2. How UNIX Is Organized On The Disk 75

Some Basic Terms 75
The Major UNIX Directories 76
Server-Client File Systems And Partitions 80
3.3. Backing Up File Systems With `dump` 81
Reel Tape vs. Cartridge Tape 81
Backing Up Diskless Clients' `ndl` Partitions 82
Sample Scripts For File System Dumps 83
File System Dumps Over Ethernet 86
Other Files Relating To `dump` 86
3.4. Restoring Files With `restore` 87
Restoring One Or More Particular Files 87
Restoring An Entire File System 88
Restoring A Damaged 'root' Or `/pub` File System 89
Restoring Files Over Ethernet 91
3.5. Maintaining File Systems 93
Disk Capacity — Checking The Disk Resource Usage 93
The Disk Quota System 94
Setting Up The Quota System 94
Some Implementation Detail 95
Administration Of The Quota System 95
The User's View Of Disk Quotas 96
What To Do When Quota Limit Is Reached 96
Quotas And The NFS 97
Making Room On File Systems 97
Files Needing Periodic Attention 98

Chapter 4 Communications **101**
4.1. The Ethernet 102
Ethernet Hardware 102
Example Of A Network Transfer 103
Configuring The Network In System Software 104
Setting Up A Gateway Machine 104
Reducing Network Overhead 107

Network Security — `/etc/hosts.equiv` and `.rhosts` 109
Special Cursor Characters During Ether Boot 111
How To Make Current Networks Compatible With Older Networks 112
Ethernet Troubleshooting 114
4.2. Wide Area Networks 117
An Overview Of `uucp` 117
Installing `uucp` 119
4.3. Setting Up The Mail Routing System 123
Picking a Name for your Domain 124
Picking a 'Main Machine' for Mail Forwarding 125
Configuring for Mail Forwarding 126
Telling Sendmail your Domain Name 126
Arpanet Forwarding 127
Setting up the 'Postmaster' Alias 127
Yellow Pages Requirements 128
Testing Your Mailer Configuration 129
Diagnosing Troubles with Mail Delivery 129
4.4. Tip 131

Chapter 5 Adding Hardware To Your System **137**
5.1. Adding A Board To Your System 138
Kernel Modification 138
File System Modification 139
Trouble Shooting 141
5.2. Connecting Devices To Asynchronous Serial Ports 142
General Theory 142
General Debugging Hints 143
5.3. Adding A Terminal To Your System 145
Serial Port and Cable Connection 145
Kernel Modification 145
File System Modification 146

Problems 148
5.4. Adding A Modem To Your System 149
Serial Ports, Cable Connection And Modem Switches 149
Kernel Modification 150
File System Modification 153
Hayes Specific Considerations 155
Problems 156
5.5. Adding A Printer To Your System 158
Hooking Up A Serial ASCII Printer 158
Serial Port and Cable Connection 158
File System Modification 159
Hooking Up A Printer To A VPC-2200 Multibus Board 162
Card Cage Installation And Cable Hook-up 162
Kernel Modification 163
File System Modification 164
Printing On Remote Machines 166
Output Filters 167
Output Filter Specifications 167
Line Printer Commands 168
lpd The Printer Daemon 168
lpc — Line Printer Control Program 169
lpr — Enter Jobs Into Print Queue 170
lpq — Show Line Printer Queue 170
lprm — Remove Jobs From A Queue 170
Access Control 170
Problems 171
Error Messages From The Line Printer System 171
lpr 171
lpq 172
lprm 173
lpc 173
lpd 173

Chapter 6 Periodic Maintenance 177
6.1. Backing Up The File System With dump 178
6.2. Bootstrap And Shutdown Procedures 180
Powering-up Self Test Procedures 180
When Critical Errors Are Found In Self Test 180
When Non-Critical Errors Are Found In Self Test 181
When No Errors Are Found In Self Test 182
The Automatic Boot Procedure 182
Booting From Specific Devices 183
Booting From Disk 184
Booting From Network Disk 184
Booting From Tape 185
Booting Files From The Default Device 185
Shutdown Procedures 186
Power Loss 186
Messages from the Monitor and the Boot Program 187
6.3. When The System Crashes 197
Crash Error Messages 197
System Core Dumps 197
Analyzing System Core Dumps 199
User Program Crashes 199
When To Call For Help 200
6.4. Kernel Configuration 201
Notes to Step 3 of Kernel Configuration 201
Note 1: Configuring Systems Without Source 201
Note 2: Adding New Device Drivers 201
Note 3: Sharing Object Modules 202
Note 4: Building Profiled Kernels 202
Rules for Defaulting System Devices 203
Configuration File Grammar 204
Lexical Conventions 206
Data Structure Sizing Rules 207
Compile Time Rules 207

Run-time Calculations 208
System Size Limitations 208
6.5. System Log Configuration 209
6.6. Monitoring System Performance 210
6.7. Resource Control 211
6.8. Accounting 212
6.9. Making Local Modifications 213
6.10. Files Needing Periodic Attention 214

Chapter 7 Upgrading System Software **217**
7.1. What To Save When Upgrading 218
7.2. How To Merge Old Files Into The New System 220

Chapter 8 Diag — A Disk Maintenance Program **225**
8.1. Architecture 225
Cylinder, Head and Sector Numbers 225
Logical vs. Physical 225
Partitions and File Systems 226
SCSI Interface 226
SMD Interface 226
Removing Bad Sectors 226
SMD Bad Sectors 226
SCSI Bad Sectors 227
8.2. Starting Diag 227
User Interface 227
Booting `diag` 228
Configuring *diag* 229
Configuring for Standard Disks 230
Other Disks 231
8.3. Preparing a New Disk 233
SCSI Disks 234
Using fix to Format SMD Disks 237
8.4. Troubleshooting With Diag 240

Checking and Fixing a Bad Sector (SCSI) 241
Checking and Fixing a Bad Sector (SMD) 243
Electronic Problems 245
8.5. Command List 246
Toggle Flags and Options 246
Miscellaneous Commands 247
Tests 247
Complicated, Interactive Commands 248
format 248
map 249
fix 249
slip 250
rhdr 250
label 252
partition 253
scan 255
whdr 256

Appendix A File System Check Program **259**
A.1. Overview of the File System 259
Superblock 259
Summary Information 260
Cylinder Groups 260
Fragments 261
Updates to the File System 261
A.2. Fixing Corrupted File Systems 262
Detecting and Correcting Corruption 262
Super-block Checking 263
Free Block Checking 263
Checking the Inode State 263
Inode Links 264
Inode Data Size 264
Checking the Data Associated with an Inode 265

File System Connectivity 265
A.3. Fsck Error Conditions 265
Conventions 265
Initialization 266
Phase 1 Check Blocks and Sizes 268
Phase 1B: Rescan for More Dup's 271
Phase 2 Check Pathnames 271
Phase 3 Check Connectivity 273
Phase 4 Check Reference Counts 274
Phase 5 - Check Cyl Groups 277
Phase 6 - Salvage Cylinder Groups 279
Cleanup 279
A.4. References 279

Appendix B UNIX File System **283**
B.1. Old File System 284
B.2. New file system organization 285
Optimizing storage utilization 286
File system parameterization 289
Layout policies 290
B.3. Performance 292
B.4. File system functional enhancements 295
Long file names 295
File locking 295
Symbolic links 297
Rename 297
Quotas 297
B.5. Software engineering 298
B.6. Acknowledgements 299
B.7. References 299

Appendix C UUCP Implementation Description **305**
C.1. UUCP — UNIX to UNIX File Copy 305

Type 1 — Local Copy 307
Type 2 — Receive Files 307
Type 3 — Send Files 307
Type 4 and Type 5 — Remote UUCP Required 308
Type 6 — Remote Execution 308
C.2. UUX — UNIX to UNIX Execution 308
User Line 309
Required File Line 309
Standard Input Line 310
Standard Output Line 310
Command Line 310
C.3. UUXQT — UUCP Command Execution 310
C.4. UUCICO — Copy In, Copy Out 310
Scan For Work 311
Call Remote System 312
Line Protocol Selection 312
Work Processing 313
Conversation Termination 314
C.5. UULOG — UUCP Log Inquiry 314
C.6. UUCLEAN — UUCP Spool Directory Cleanup 314
C.7. Security 315
C.8. UUCP Installation 315
uucp.h Modification 316
Makefile Modification 316
Compiling the System 317
Files Required for Execution 317
L-devices — Call Unit Information File 317
L-dialcodes — Dial Code Abbreviations File 318
USERFILE 318
L.sys 320
L.cmds 322
Device Types 322
C.9. Administration 323

SQFILE — Sequence Check File 323
TM — Temporary Data Files 323
LOG — Log Entry Files 323
STST — System Status Files 324
LCK — Lock Files 324
Shell Files 324
Login Entry 325
File Modes 325

Appendix D Sendmail Installation and Operation **329**
D.1. Basic Installation 329
Off-the-Shelf Configurations 330
Installation Using the Makefile 330
Installation by Hand 330
/usr/lib/sendmail 330
/usr/lib/sendmail.cf 330
/usr/spool/mqueue 330
/usr/lib/aliases* 330
/etc/rc 331
/usr/lib/sendmail.fc 331
/usr/lib/sendmail.hf 331
/usr/lib/sendmail.st 331
/usr/etc/in.syslog 331
D.2. Normal Operations 332
Quick Configuration Startup 332
The System Log 332
Format 332
Levels 332
The Mail Queue 332
Printing the Queue 332
Format of Queue Files 333
Forcing the Queue 334
The Local Alias Database 335

Rebuilding the Alias Database 335
Potential Problems 335
List Owners 336
Yellow Pages Aliases 336
Naming Conventions 336
Potential Problems 336
Per-User Forwarding (.forward Files) 337
Special Header Lines 337
Return-Receipt-To: 337
Errors-To: 337
To: 337
D.3. Arguments 337
Queue Interval 337
Daemon Mode 337
Forcing the Queue 338
Debugging 338
Trying a Different Configuration File 338
Changing the Values of Options 338
D.4. Tuning 338
Timeouts 339
Queue Interval 339
Read Timeouts 339
Message Timeouts 339
Delivery Mode 339
Load Limiting 340
Log Level 340
File Modes 341
To Suid or not to Suid? 341
Temporary File Modes 341
Should my Alias Database be Writable? 341
D.5. The Whole Scoop on the Configuration File 341
The Syntax 342
R and S — Rewriting Rules 342

D — Define Macro 342
C and F — Define Classes 342
M — Define Mailer 343
H — Define Header 343
O — Set Option 343
T — Define Trusted Users 344
P — Precedence Definitions 344
The Semantics 344
Special Macros, Conditionals 344
Special Classes 346
The Left Hand Side 346
The Right Hand Side 347
Semantics of Rewriting Rule Sets 347
Mailer Flags, etc. 349
The ''error'' Mailer 349
Semantics of Mailer Descriptions 349
Building a Configuration File from Scratch 351
What you are Trying to do 351
Philosophy 351
Relevant Issues 353
How to Proceed 353
Testing the Rewriting Rules — the bt Flag 353
D.6. Command Line Flags 355
D.7. Configuration Options 357
D.8. Mailer Flags 359
D.9. Summary of Support Files 361

Appendix E sendmail — Internetwork Mail Router **365**
E.1. Design Goals 366
E.2. Overview 368
System Organization 368
Interfaces to the Outside World 368
Argument vector/exit status 368

SMTP over Pipes 368
SMTP over an IPC Connection 368
Operational Description 368
Argument Processing and Address Parsing 368
Message Collection 369
Message Delivery 369
Queueing for Retransmission 369
Return to Sender 369
Message Header Editing 369
Configuration File 370
E.3. Usage and Implementation 370
Arguments 370
Mail to Files and Programs 370
Aliasing, Forwarding, Inclusion 371
Aliasing 371
Forwarding 371
Inclusion 371
Message Collection 371
Message Delivery 372
Queued Messages 372
Configuration 372
Macros 372
Header Declarations 373
Mailer Declarations 373
Address Rewriting Rules 373
Option Setting 373
E.4. Evaluations and Future Plans 373

Tables

Table 2-1 Sectors/Track With Xylogics Controller 15
Table 2-2 Sectors/Track With Adaptec Controller 15

Table 3-1 Sample of Fujitsu 130 MB Disk Partitions 73

Table 5-1 Sun Supported Boards 140
Table 5-2 Using `MAKEDEV` For New Boards 146

Table 6-1 Files Needing Periodic Attention 214

Table 7-1 Standard List of Files to Save when Upgrading 218
Table 7-2 UNIX Tape Device Abbreviations 219

Table 8-1 Controller Bus Addresses 230
Table 8-2 Default Partition Sizes for SCSI Disk Subsystems 237
Table 8-3 Default Partition Sizes for SMD Disk Subsystems 240

Table B-1 Wasted Space as a function of Block Size 287
Table B-2 Reading Rates of the Old and New UNIX File Systems 293
Table B-3 Writing rates of the old and new UNIX file systems 293

Figures

Figure 4-1 A Local Network 103
Figure 4-2 Two Local Networks Joined By A Gateway Machine 105

Figure 5-1 Null Modem 142
Figure 5-2 Communications Through Modems 143

Figure B-1 Example layout of blocks and fragments in a 4096/1024 file system 287

Figure D-1 Rewriting Set Semantics 348

Figure E-1 Sendmail System Structure 367

System Administration *for the* Sun Workstation

1

Introduction And General Advice

Introduction And General Advice 3

1

Introduction And General Advice

This *System Administration Manual* explains many different procedures and types of maintenance that need to be done to keep your Sun Workstation and your Sun network up and running smoothly. It also tells you how to change the size or configuration of your system, and how to expand your system, when those are your goals. In addition, it tells you how to restore a system to its former state when disaster occurs (as it unfortunately does sometimes on all operating systems). Many of the procedures covered in this manual can and should be done on every system, even the least active standalone system. However, there are things we cover that you may never need to do at your particular site. Use the manual as a sort of almanac for consultation and verification; it gives exact procedures for many aspects of system administration.

The manual often refers you to some other manual in the Sun user documentation. Wherever possible we try to describe entire procedures and give complete examples, however it is not possible to simply reproduce all the information that is given in other sections of Sun's documentation. As you gain familiarity with this manual and with the Sun system, you will more and more be able to rely on the terse explanations that cover some long processes.

Always read sections completely before jumping in

Always read through an entire section when you are doing any procedure for the first time. Never try to leap ahead of the given instructions unless you are absolutely certain of your moves, and willing to suffer any mistakes you make. The machine has no patience, but it is not in any hurry either.

Developing Your Administration Procedures

As system administrator, you can make your work easier and get problems fixed more quickly by following the suggestions below; most of them are simple common sense. Remember, each UNIX† site should develop administrative procedures and standards to fit its situation — we encourage you to do this, it helps prevent rash action in a puzzling or unexpected situation. There are several good introductory UNIX books on the market. If you are a novice get one of these to read and keep around the machine room. Here is our introductory checklist for new systems — it covers the most obvious things. You may add your own procedures and ideas to it.

† UNIX is a trademark of AT&T Bell Laboratories.

- Keep a notebook describing the layout of the system, including a history of any changes you have made. In particular you should save hardcopy records of the file `/etc/nd.local` and of your disk label(s). This manual tells you what these are and where to find them.
- If you are not experienced with the Sun environment, start with a 'vanilla' system; customize it only after you've gained experience and some expertise.
- Make dump tapes regularly. Tech Support occasionally gets calls from people who did not make dump tapes, and have just accidentally removed crucial files from their disk. Without dump tapes, lost files are gone forever. This manual describes how to make dump tapes.
- If you are going to make a major change to a file system, do full dumps first. This is in addition to the routine dumps that you should be doing regularly.
- Plan any changes completely before implementing them. If you forget a step or do something out of order, you might introduce big problems.
- If you run into trouble and are not sure what to do, call Tech Support. If your system is out of the warranty period, you will be charged for the call unless you have a support contract. We strongly urge you to get a support contract. Contact your local Sun sales office for details.

2

Sun Network Services

Sun Network Services 7

2.1. Introduction And Terminology 7
Networking Models 7
Terminology 8
UNIX Meets Sun Network Services 8
A Hint About Debugging UNIX In The Network Environment 9
2.2. nd: The Sun Network Disk Service 10
Partitions 10
The /etc/nd.local File 10
The Server's View Of Partitions 13
The Client's View Of Partitions 13
Using Clients' Root Partitions from the Server 13
Using Clients' Swap Partitions From The Server 16
More Advanced Use 16
Words of Warning 17
2.3. NFS: The Network File System Service 17
What Is The NFS Service 17
How The NFS Works 18
How To Become An NFS Server 18
How To Remote Mount A File System 19
Typical NFS Layout 19
Debugging the Network File System 22
Incompatibilities with Earlier Versions 30

Beefing Up NFS Security 33
Clock Skew In User Programs 34
2.4. yp: The Sun Yellow Pages Service 35
What Is The Yellow Pages Service? 35
Yellow Pages Overview 36
Yellow Pages Installation and Administration 38
Debugging A Yellow Pages Client 46
Debugging A Yellow Pages Server 49
Yellow Pages Policies 52
How Security Is Changed With The Yellow Pages 52
What If You Do Not Use The Yellow Pages? 56
2.5. Adding A New User To A Machine 56
Edit The Master yp Server's /etc/passwd File 56
Make A Home Directory 58
The New User's Environment 58
2.6. Adding A Client Machine To An nd Server 59
Choose A Unique Client Name 59
Edit Two Files On The Master yp Server 59
Edit The nd Server's /etc/nd.local File 60
Preparing An Unused Partition For A New Client 62
Preparing A Previously Used Client Partition 64
Booting The New Client Machine 65

2

Sun Network Services

2.1. Introduction And Terminology

This chapter introduces the Sun Network services. We describe the services currently available, and define some terms in the network environment.

Following that, we introduce and explain each of the three types of service now available for the Sun UNIX workstation: network disk service, network file system service, and yellow pages service. Within each of the three sections you will find information about periodic maintenance and trouble-shooting for the service under discussion.

While some of this material tends to be theoretical, its specific implications will be seen again and again as you become familiar with system administration. For example, if you run the yellow pages, you must understand that some typical UNIX procedures have changed in the yellow pages environment. That is also true if you use the network disk or the network file system. This chapter covers only those aspects of network services necessary for performing the duties of system administration. For a complete theoretical overview, see *The Network Services Guide*.

At the end of this chapter we explain how to add new users and new client machines at your site, one of the first and most important duties of system administration.

Networking Models

There are many ways to make computers and networks interface transparently. The two major ones are the distributed operating system approach and the network services approach.

A distributed operating system allows the network software designer to make grand assumptions about the other machines on the network. Usually two of these assumptions are: that the remote piece of hardware is identical to the local hardware, and that the remote and local machines are running identical software. These assumptions allow a quick and simple implementation of a network system in an environment limited to specific hardware and software. This type of distributed operating system is, by design, closed. That is to say, it's very difficult to integrate new hardware or software into a closed network environment, unless it comes from the vendor of that network system. A closed network system forces a customer to return to one vendor for solutions to all computing needs.

On the other hand, the Sun network is not closed. Sun bases its networking on network services. Network services are made up of remote programs composed of remote procedures called from the network. Optimally, a remote procedure computes results based entirely on its own parameters. Thus, the procedure (and therefore, the network service) is not tied to any particular operating system or hardware. The design of the Sun network makes it possible for a variety of machines and software to talk to the network services, enabling the Sun Workstation to talk to various types of computers.

As you might expect, the Sun network services are more complex in design and implementation than a closed distributed operating system. Since remote procedures are operating-system independent and hardware independent, multiple remote procedures must sometimes be called in the Sun environment, where a single transaction might suffice in a closed system.

Terminology

The bulk of this chapter explains administration of three Sun network services — the Network Disk (`nd`), the Sun Network File System (NFS), and the Sun Yellow Pages (`yp`). Before discussing these three services, we define some generic concepts and terms.

Server

A *server* is any machine that provides one of the three network services. A single machine may provide more than one service. In fact, a typical configuration would be for one machine to act as an `nd`-server, an NFS-server, and a `yp`-server. In each of the Sun network services, servers are entirely passive. The servers wait for clients to call them, they never call the clients.

Client

A *client* is any entity that accesses a network service. We use the term entity because the thing doing the accessing may be an actual machine or simply a UNIX process generated by a piece of software.

The degree to which clients are bound to their server varies with each of the Sun network services. For example, in `nd` service the `nd` client is strictly bound to the `nd` server because the server supplies a private area of a large disk for the exclusive use of each client. At the other extreme, a Sun `yp` client binds randomly to one of the `yp` servers by broadcasting a request. At any point, the `yp` client may decide to broadcast for a new server. The Sun NFS stands somewhere in the middle. An NFS client may choose to mount file systems from any number of servers at any time.

In all cases, however, the client initiates the binding. The server completes the binding subject to access control rules specific to each service. Since most network administration problems occur at bind time, a system administrator should know how a client binds to a server and what (if any) access control policy each server uses.

UNIX Meets Sun Network Services

Unlike many recently marketed distributed operating systems, UNIX was originally designed without knowledge that networks existed. This 'networking ignorance' presents three impediments to linking UNIX with currently available high performance networks:

1) UNIX was never designed to yield to a higher authority (like a network authentication server) for critical information or services. As a result some UNIX semantics are hard to maintain 'over the net.' For example, trusting user id 0 (root) is not always a good idea.

2) Some UNIX execution semantics are difficult. For example, UNIX allows a user to remove an open file, yet the file does not disappear until closed by everyone. In a network environment a client UNIX machine may not own an open file. Therefore, a server may remove a client's open file.

3) When a UNIX machine crashes, it takes all its applications down with it. When a network node crashes (whether client or server), it should not drag all of its bound neighbors down. The treatment of node failure on a network raises difficulties in any system and is especially difficult in the UNIX environment. Sun has implemented a system of 'stateless' protocols to circumvent the problem of a crashing server dragging down its bound clients. Stateless here means that a client is independently responsible for completing work, and that a server need not remember anything from one call to the next. In other words, the server keeps no state. With no state left on the server, there is no state to recover when the server crashes and comes back up. So, from the client's point of view, a crashed server appears no different than a very slow server.

In implementing UNIX over the network, Sun attempted to remain compatible with UNIX whenever possible. However, certain incompatibilities have been introduced; they are typically of two kinds. First, those issues that would make a networked UNIX evolve into a distributed operating system, rather than a collection of network services. And second, those issues that would make crash recovery extremely difficult from both the implementation and administration point of view.

All incompatibilities are documented in the appropriate sections of this administration manual.

A Hint About Debugging UNIX In The Network Environment

When you cannot get something done that involves Sun network services, the problem probably lies in one of the following four areas. They are listed here with the most likely problem first.

1) The Sun network access control policies do not allow the operation, or architectural constraints prevent the operation.

2) The client software or environment is broken.

3) The server software or environment is broken.

4) The network is broken.

The following sections present specific instructions on how to check for these causes of failure in the nd, NFS, and yp environments.

2.2. nd: The Sun Network Disk Service

This section describes network disks, the `/etc/nd` command, the `/etc/nd.local` file, and the relevant `/dev` entries. See nd(4P) in the *System Interface Manual* and nd(8) in the *Commands Reference Manual.*

Network disks pertain only to file servers and clients, not to stand-alone systems.

Partitions

In the typical `nd` environment, a public partition is shared between the disk server and all the clients. Also, each client has private root and swap partitions on the server's disk(s).

There are two types of disk partitions — hard and soft. Hard partitions are defined on the disk label, written onto the disk by the `diag` utility. By default, `diag` divides an `nd` server's disk into partitions `a`, `b`, `d`, `e`, and `g`. Partition `a` is the server's root partition, partition `b` is the server's swap partition, partition `d` is the `/usr` partition, partition `e` is the `/usr2` partition, and partition `g` is the rest of the disk.

In addition, there is typically a `c` partition which includes and provides access to the entire physical disk. It is occasionally used to store a single large file system or to access the entire disk.

On a server, partition `g` is further partitioned by the `/etc/nd` utility into several soft partitions. A single command may be given to `/etc/nd` on the command line; typically, however, the file `/etc/nd.local` is used as input; it defines the soft partitions produced by `/etc/nd`. To run `/etc/nd` with input from this file type the following:

```
# /etc/nd < /etc/nd.local
```

An `/etc/nd` command line like this appears in the `/etc/rc.local` script, which executes when the server goes from single-user to multi-user. If you boot single-user, the `/etc/nd` utility is not executed — unless you type the above command explicitly.

The /etc/nd.local File

Lines in the file beginning with '#' (sharp) are comments. Other lines in the file form a list of commands to the `/etc/nd` utility. There are six commands available: `user`, `version`, `soff`, `son`, `clear`, and `serverat`.

Below is an example of an `/etc/nd.local` file on a server called `venus`, followed by an explanation of the contents.

```
#       venus
clear
version 1
#
user 0 0 /dev/xy0g 0 11960 -1
user bill 0 /dev/xy0g 11960 10120 0
user bill 1 /dev/xy0g 22080 20240 -1
user debby 0 /dev/xy0g 42320 10120 1
user debby 1 /dev/xy0g 52440 20240 -1
#The following are on an entirely separate
#device, /dev/xy2h
user joan 0 /dev/xy2h 0 10120 2
user joan 1 /dev/xy2h 10120 20240 -1
user mike 0 /dev/xy2h 30360 10120 3
user mike 1 /dev/xy2h 40480 20240 -1
son
```

The commands and their variables are explained below.

- **user** *client_name unit# device startblk nblks ndl#*

Where the variables have the following meaning:

client_name — The name of the client owning this partition. Incoming requests from *client_name* at *unit#* are transformed into server device *device* at the location addressed by *startblk* and *nblks*. If *client_name* is '0', this is a public partition. The special clientname 'localhost' is conventionally used for spare or locally used nd partitions.

unit# — The unit number of the partition. When a *client_name* is present, a '0' (zero) refers to the client's root partition, and a '1' refers to the client's swap partition. When the *client_name* is a '0', *unit#* refers to a public unit.

device — The `/dev` entry corresponding to the hard partition that this soft partition is on. Typically, this is `/dev/xy0g` on a Xylogics SMD controller, or `/dev/sd0g` on an Adaptec SCSI controller.

startblk — The offset from the beginning of *device* to the beginning of this soft partition. The offset is in 512 byte units.

nblks — The size of this partition in 512 byte units. If *nblks* is '-1' then this *unit#* is equivalent to the entire partition *device*; no soft partitioning of the hard partition is done.

ndl# — The `/dev/ndl*` entry corresponding to this partition, where '*' corresponds to `ndl#` on this line. There is no corresponding `/dev/ndl*` entry when `ndl#` is '-1'; this is usually the case with a swap partition or a soft partition which starts at the beginning of a hard partition — like the public partition

/dev/xy0g above. Swap partitions do not contain file systems; therefore they are never used as arguments to fsck, df, mount, dump, or other similar utilities. Soft partitions situated at the start of a hard partition may be referenced by the hard partition device name — for example, /dev/xy0g for the public partition defined above. All other soft partitions containing file systems can only be referenced by their /dev/ndl# device name. Note that there is a limit of twenty soft partitions on a system, so this number must be between 0 and 19.

- **version** *version_number*
  The version command line gives the level of the system configuration of the server. This number is sent by the server to the client when the client boots, and the client must send it back for every network disk transaction. If the server's version has been changed, his kernel table will have changed and the server will no longer respond to clients who have not booted under the latest server version. This is one way to insure that all clients reboot after the installation of a new kernel. However, the best way to insure that clients reboot is to halt them, using /etc/halt, before making any server modifications which will affect clients.

- **soff**
  Stops the network disk service until a subsequent **son** command.

- **son**
  Starts the network disk service. This command should be issued at the end of /etc/nd.local, after all user, and version commands.

- **clear**
  Stops the network disk service and clears all user and ether information.

- **serverat** *server_name*
  If you have your own disk you may use serverat to specify a network disk in addition to your own. However, this command is necessary only if you wish to use a public network disk or change network disk servers.

  For example, suppose omar, a machine with a local disk, wants to mount the /pub partition of venus, a network disk server on the same local network. To do this become superuser on omar and do the following:

  1) Create the device node /dev/rndl0. This is required to run the nd utility. The command below creates both the block, /dev/ndl0, and the raw, /dev/rndl0, devices.

```
# cd /dev
# MAKEDEV ndl0
```

  2) If omar is running a non-generic kernel, make sure it includes the line: pseudo-device nd. Reconfigure the kernel if you change it.

  3) Create the /pub directory:

```
# mkdir /pub
```

4) Add the following lines to the end of `/etc/rc.local`:

```
/etc/nd serverat venus
/etc/mount -r /dev/ndp0 /pub &
```

Note that the mount command must specify the `-r` (read-only) when a client mounts a public partition from a server.

Next time `omar` reboots it will mount `venus`'s public partition **0** (zero) on `/pub`. The mount command is run in the background so that `omar` can still be booted if `venus` is down. However, if `venus` goes down while `omar` is using files in `/pub`, processes on `omar` may hang until `venus` comes up again.

The Server's View Of Partitions

The public partition is the first soft partition defined by `/etc/nd` on the hard partition `g`. It is referenced by the name of the hard partition, typically `/dev/xy0g`. Following the public partition are the clients' root and swap partitions. The client root partitions are referenced by `/dev/nd1*`, where the '*' is an integer number. The appropriate device for each client is determined by the `nd1#` field in `/etc/nd.local`. For example, looking again at a sample `/etc/nd.local` file entry, we see the root partition for `joan` is `/dev/nd12` (network disk local two).

```
user joan 0 /dev/xy2h 0 10120 2
user joan 1 /dev/xy2h 10120 20240 -1
```

The clients' swap partitions cannot ordinarily be referenced from the server.

See below for discussion of how to use client's root and, in special cases, swap partitions from the server.

The Client's View Of Partitions

From a client, the public partition is referenced through `/dev/ndp0` (network disk public zero). The root partition is referenced through `/dev/nd0` (network disk zero). The swap partition is referenced through `/dev/nd1`. The swap partition does not have a file system on it and should never be mounted.

Using Clients' Root Partitions from the Server

A server can access the file system on a client's root partition in the following way. First, halt the client; this **must** be done or you risk corrupting the file system. On the server, look in the `/etc/nd.local` file for the client's `nd1` number. Now mount the desired partition with the following command:

```
# /etc/mount /dev/nd11 /mnt
```

If this were done on the system described in the sample file above, it would mount the root partition of client `debby` on the `/mnt` directory of the server. Mounting in this way would then permit access to and modification of `debby`'s

intact file system by the server. With caution, you can also use the command:

```
# /etc/mount -r /dev/nd11 /mnt
```

Which mounts the file system read-only, and the client need not be halted. Use extreme caution when mounting a client that is not halted — if the client partition is mounted writeable and the client is still running you will destroy the client's file system!

These procedures allow the server, or with `-r` the server and the client, to access the files on the client's root partition. The paragraphs below describe how to use a client's root partition to create more file space on a system. (Note that the next section of this chapter describes how to use a client's swap space for the same purpose. If the swap space is large enough for your task, you may use it without having to back up files as described below for root partitions.)

Occasionally, the server needs to write on a client's soft partition space. Be advised that this will wipe out everything on the soft partition; therefore, the partition should be dumped to tape first, or determined to be expendable. After the dump, make sure the client is halted or powered-off. Then, use the `/etc/mkfs` command to make a file system on the target partition. Unfortunately, the `/etc/newfs` command cannot be used on soft partitions. The `mkfs` command is:

```
# /etc/mkfs device nblks sectors heads blksize fragsize
```

Where the variables have the following meanings:

device	the `/dev` entry for the soft partition.
nblks	the size of the partition in 512 byte units. This value is obtained from the *nblks* field in `/etc/nd.local`.
sectors	the number of sectors/track on the physical disk.
heads	the number of heads (i.e., tracks/cylinder).
blksize	the size of a block on the new file system. Use 8192.
fragsize	the size of a fragment on the new file system. Use 1024.

The *sectors* and *heads* values are dependent upon the type of physical disk and controller.

The values for current disks with a Xylogics 450 controller are:

Table 2-1 *Sectors/Track With Xylogics Controller*

Disk Type	Sectors	Heads
D84	32	7
D168	32	10
D169	32	10
Eagle	46	20

The values for current disks with an Adaptec SCSI controller are:

Table 2-2 *Sectors/Track With Adaptec Controller*

Disk Type	Sectors	Heads
Micropolis	17	6

To get these values for any disk or controller use the `/etc/dkinfo` command. Use the proper abbreviation for controller type — `sd` for Adaptec SCSI controller and `xy` for Xylogics SMD controller — and the proper unit number that you want the information for. For a client with a Xylogics controller, unit number 0 (zero), the command would be:

```
# /etc/dkinfo xy0
```

Returning to the hypothetical system described above (in the section *The /etc/nd.local File*), let us assume a Xylogics 450 disk controller and an Eagle disk. The `/etc/mkfs` command for `bill`'s root partition is shown below. Remember, this will destroy all data on client `bill`!

```
# /etc/mkfs /dev/nd10 10120 46 20 8192 1024
```

After making a file system, the partition can be mounted and used. Typically, you want to run `/etc/fsck` on the partition and mount it as part of the boot procedure. The first reaction is to put an entry in `/etc/fstab` to do just that, however, `/etc/fsck` runs before `/etc/nd` in the boot procedure so the soft partition is not defined when `/etc/fsck` is run. It is inadvisable to run `/etc/nd` before `/etc/fsck` because that makes the public partition available to clients before its file system has been checked.

The correct approach is to add commands to the end of `/etc/rc.local`. Continuing the example from above, to make `bill`'s root partition available for use add the following lines to the end of the server's `/etc/rc.local` file:

```
echo Fsck on the /dev/ndl0 disk  > /dev/console
/etc/fsck -p /dev/ndl0  > /dev/console
/etc/mount /dev/ndl0 /spare
```

The `/etc/fsck` takes a little while so the `echo` command is a reminder of what is happening. Without it, people often think the boot procedure has hung. The `/etc/mount` makes the disk accessible on the directory */spare* that you have created for that purpose. Of course, you may give the */spare* directory any name.

Using Clients' Swap Partitions From The Server

To use a client's swap partition space from the server, it must first be given a valid `ndl` number. Remember, swap partitions are initially given an `ndl` number of '-1' on the `user` line of the `nd.local` file. First, comment out the old line you wish to change — that is, precede it with a # (sharp) so the system will ignore it, but leave it in the file so you can easily change back at a later date if desired. Then, add a line to the file that changes the targeted entry to a number which has not been assigned to any other partition. (Note: only 0-19 are valid `ndl` numbers.) Any time the `/etc/nd.local` file is changed, `/etc/nd` must be run again to make the change take effect. Thus after editing you must type:

```
# /etc/nd < /etc/nd.local
```

Then the swap partition may be used by following the procedure given above for client root partitions, but without the need to back up files.

More Advanced Use

You may have more than one public partition, and/or more than one private, writable client partition.

`/etc/nd.local` describes a public partition with a line of the form:

```
user 0 0 /dev. . .
```

The first 'zero' indicates the partition is public and the second 'zero' indicates the public partition unit number. Therefore, a second public partition is created by adding the following line to `/etc/nd.local`

```
user 0 1 /dev. . .
```

The clients reference this second public partition using `/dev/ndp1`.

A second writable partition is assigned to a client in a similar manner. The line

```
user joan 2 /dev. . .
```

in the `/etc/nd.local` file makes partition '2' available to `joan` through `/dev/nd2`.

Remember to run /etc/nd each time you change the /etc/nd.local file. Halt any affected clients before a server changes (using /etc/halt), and reboot clients after completing the change.

Words of Warning

The /etc/nd command does very little error checking. It does not check for overlapping partitions, multiple entries for the same user, unreasonable starting points, or unreasonable partition sizes.

When /etc/nd does find an error it prints a message saying the line was ignored. That usually means /etc/nd stopped processing at that point. Sometimes it does go on, but in that case gets the whole rest of the file wrong. In either case, the line ignored error message means you must fix the problem and run /etc/nd again.

Tabs and blank lines are not allowed in the /etc/nd.local file.

2.3. NFS: The Network File System Service

We begin with an explanation of some NFS terms and concepts. Then we describe how to create an NFS server that exports file systems, and how to mount and utilize remote file systems. Following that a section on debugging the NFS explains what to do when problems occur. Finally, some cautionary words about incompatibilities between NFS files and normal UNIX files.

What Is The NFS Service

In its ability to allow many clients simultaneous access to a single file system, the NFS is quite different from nd (Network Disk). With nd, each client 'owns' a fixed part of a disk partitioned for client use on an nd server. With NFS diskless clients still have a permanent bond to their nd server, but may also mount (and unmount) many other file systems when they choose. Thus, while a client has only one nd server, it may have many NFS servers.

The NFS enables users to share file systems over the network. A client may mount or unmount file systems from an NFS server machine. The client always initiates the binding to a server's file system by using the mount(8) command. Typically, a client mounts one or more remote file systems at startup time by placing lines like these in the file /etc/fstab, which mount reads when the system comes up:

```
titan:/usr2 /usr2 nfs rw,hard 0 0
venus:/usr/man /usr/man nfs rw,hard 0 0
```

See fstab(5) for a full description of the format.

Since clients initiate all remote mounts, NFS servers keep control over who may mount a file system by limiting named file systems to desired clients with an entry in the /etc/exports file. For example:

```
/usr/local                    # export to the world
/usr2      nixon ford reagan # export to only these machines
```

is an /etc/exports entry that speaks for itself. Note that pathnames given in /etc/exports must be the mount point of a local file system. See

exports(5) for a full description of the format.

How The NFS Works

Two remote programs implement the NFS service — mountd(8) and nfsd(8). A client's mount request talks to mountd which checks the access permission of the client and returns a pointer to a filesystem. After the mount completes, access to that mount point and below goes through the pointer to the server's nfsd daemon using rpc(4). Client kernel file access requests (delayed-write and read-ahead) are handled by the biod(8) daemons on the client.

If this terse explanation doesn't answer all your questions, you can find more details in *The Network Services Guide*. However, you will be able to install and administer the NFS just by considering the information given here.

How To Become An NFS Server

An NFS server is simply a machine that exports a file system or systems. Even a diskless nd client may become an NFS server. Here are the three steps that any machine must follow to enable it to export a file system.

1) Become super-user and place the mount-point pathname of the file system you want to export in the file /etc/exports. See exports(5) for file format details. For example, to export /usr/src/mybin, the export file would look like:

```
/usr/src/mybin
```

Of course, an NFS server may only export file systems of its own.

2) As we saw above, mountd must be present for a remote mount to succeed. Make sure mountd will be available for an rpc call by checking the file /etc/servers, on the NFS server, for this line:

```
rpc     udp  /usr/etc/rpc.mountd 100005 1
```

If it isn't there add it. For details, see servers(5).

3) Remote mount also needs some number (typically 4) of nfsd's to execute on NFS servers. Check /etc/rc.local for lines like these:

```
if [ -f /etc/nfsd -a -f /etc/exports ]; then
/etc/nfsd 4 & echo -n ' nfsd'       >/dev/console
```

Add these lines, or your own version of them, if the new NFS server's /etc/rc.local does not enable nfsd's. You can enable nfsd's without rebooting, by typing, while super-user:

```
# /etc/nfsd 4
```

After these steps, the NFS server should be able to export the named file system. Read the next section and try to remote mount.

How To Remote Mount A File System

You can mount any exported file system onto your machine, as long as you can reach its server over the network, and you are included in its `/etc/export` list for that file system. On the machine where you want to mount the file system, become super-user and type the following:

```
# mount server_name :/file_system /mount_point
```

For example, to mount the manual pages from remote machine `elvis` onto the local empty directory `/usr/man`:

```
# mount -o soft elvis:/usr/man /usr/man
```

Things like manual pages are usually mounted soft, so that if `elvis` goes down, it doesn't drag down the local machine. To make sure you have mounted a file system, and mounted it where you expected to, use either `df`(1) or `mount`(8), without an argument. Each of these displays the currently mounted file systems.

Typically, you mount frequently used file systems at startup by placing an entry for them in the file `/etc/fstab`. If you are a diskless client you can look at the `/etc/fstab` for examples. You can also see `fstab`(5).

Typical NFS Layout

To demonstrate the layout of the NFS on diskless clients, the output from `mount` commands below shows the mounted file systems on a server, and on one of its clients — notice where the client file systems are mounted from. Following that, the output from `ls` commands shows the contents of various directories on the client machine. `lenin` is a diskless client of `marx`:

```
marx% mount
/dev/xy0a on / type 4.2 (rw)
/dev/xy0e on /pub.MC68010 type 4.2 (rw)
/dev/xy0g on /usr.MC68010 type 4.2 (rw)
/dev/xy2g on /usr.MC68010/marx type 4.2 (rw)
```

```
lenin% mount
/dev/nd0 on / type 4.2 (rw)
/dev/ndp0 on /pub type 4.2 (ro)
marx:/usr.MC68010 on /usr type nfs (ro,hard)
marx:/usr.MC68010/marx on /usr/marx type nfs (rw,hard)
```

```
lenin% ls -l /
total 60
lrwxrwxrwx  1 root            7 Oct  7 13:57 bin -> pub/bin
-rwxr-xr-x  1 root        38225 Oct  5 21:55 boot
drwxrwxr-x  2 root         2048 Oct 10 03:01 dev
drwxrwxr-x  3 root         1536 Oct 10 03:01 etc
lrwxrwxrwx  1 root            7 Oct  7 13:57 lib -> pub/lib
drwxr-xr-x  2 root         8192 Oct  6 17:04 lost+found
drwxrwxr-x  2 root           24 Oct  6 17:07 mnt
lrwxrwxrwx  1 root           15 Oct  7 13:57 private -> private.MC68010
drwxrwxr-x  3 root          512 Oct  6 17:08 private.MC68010
drwxr-xr-x  6 root          512 Oct  7 13:58 pub
lrwxrwxrwx  1 root            9 Oct  7 13:57 stand -> pub/stand
drwxrwxrwx  2 root          512 Oct 10 14:46 tmp
drwxr-xr-x 49 root         1024 Oct 10 10:49 usr
lrwxrwxrwx  1 root           10 Oct  7 13:57 vmunix -> pub/vmunix
```

```
lenin% ls -l /pub
total 1050
drwxrwxrwx  2 root         1044 Oct  7 15:34 bin
-rwxr-xr-x  1 root        38225 Oct  1 13:16 boot
drwxr-xr-x  2 bin           512 Oct  7 11:26 lib
drwxr-xr-x  2 root         8192 Oct  7 11:17 lost+found
drwxrwxrwx  2 root          512 Oct  7 11:27 stand
-rwxr-xr-x  1 root       460078 Oct  6 22:33 vmunix
```

```
lenin% ls -l /usr
lrwxrwxrwx  1 root           24 Oct  7 12:20 adm -> /private.MC68010/usr/adm
drwxr-xr-x  2 bin          2560 Oct  9 14:45 bin
lrwxrwxrwx  1 root           17 Oct  9 14:44 crash -> /usr/marx/crash
drwxr-xr-x  2 root           24 Oct  8 21:29 demo
drwxr-xr-x  3 bin           512 Oct  7 12:17 dict
drwxr-xr-x  2 root           24 Oct  8 21:04 doctools
drwxr-xr-x  4 bin          1536 Oct  7 12:13 etc
drwxr-xr-x  2 bin          6144 Oct  7 18:33 hosts
drwxr-xr-x 25 bin          1536 Oct  7 12:47 include
drwxr-xr-x 17 bin          2560 Oct  7 18:33 lib
drwxrwxrwx  7 root         1024 Oct  7 18:38 local
drwxr-xr-x  2 root         8192 Oct  7 11:17 lost+found
drwxrwxrwx  2 root           24 Oct  7 12:20 man
drwxrwxr-x  2 root          512 Oct  7 12:18 mdec
lrwxrwxrwx  1 root           29 Oct  7 12:20 preserve -> /private.MC68010/usr/preserve
drwxr-xr-x  2 bin           512 Oct  7 12:17 pub
drwxr-xr-x  3 bin           512 Oct  7 12:18 sccs
lrwxrwxrwx  1 root           26 Oct  7 12:20 spool -> /private.MC68010/usr/spool
drwxrwxr-x 22 root          512 Oct  7 23:51 sys
lrwxrwxrwx  1 root           24 Oct  7 12:20 tmp -> /private.MC68010/usr/tmp
drwxr-xr-x  2 bin          1536 Oct  7 12:40 ucb
```

```
lenin% ls -l /usr/lib | grep " ->"
lrwxrwxrwx  1 root           32 Oct  7 12:20 aliases -> /private.MC68010/usr/lib/aliases
lrwxrwxrwx  1 root           36 Oct  7 12:20 aliases.dir -> /private.MC68010/usr/lib/aliases.dir
lrwxrwxrwx  1 root           36 Oct  7 12:20 aliases.pag -> /private.MC68010/usr/lib/aliases.pag
lrwxrwxrwx  1 root           32 Oct  7 12:20 crontab -> /private.MC68010/usr/lib/crontab
lrwxrwxrwx  1 root           19 Oct  7 18:33 news -> /private.MC68010/usr/lib/news
lrwxrwxrwx  1 root           36 Oct  7 12:20 sendmail.cf -> /private.MC68010/usr/lib/sendmail.cf
lrwxrwxrwx  1 root           29 Oct  7 12:20 uucp -> /private.MC68010/usr/lib/uucp
```

```
lenin% ls -l /private/usr
total 5
drwxrwxr-x  2 root          512 Oct  6 21:45 adm
drwxrwxr-x  2 root          512 Oct  6 20:31 lib
drwxrwxr-x  2 root           24 Oct  6 17:08 preserve
drwxrwxr-x 13 root          512 Oct 10 03:01 spool
drwxrwxrwx  2 root          512 Oct 10 13:09 tmp
```

```
lenin% ls -l /private/usr/lib
total 25
-rw-r--r--  1 root          1151 Oct  6 20:31 aliases
-rw-r--r--  1 root             0 Oct  6 20:31 aliases.dir
-rw-r--r--  1 root          1024 Oct  6 20:31 aliases.pag
-rw-r--r--  1 root           486 Oct  6 20:31 crontab
-r--r--r--  1 root         11813 Oct  6 20:23 sendmail.cf
```

Debugging the Network File System

Before trying to debug the NFS read the section on how the NFS works and also the man pages for `mount`(8), `nfsd`(8), `biod`(8), `showmount`(8), `rpcinfo`(8), `mountd`(8), `inetd`(8), `fstab`(5), `mtab`(5) and `exports`(5). You don't have to understand them fully, but you should be familiar with the names and functions of the various daemons and database files.

When tracking down an NFS problem keep in mind that, like all network services, there are three main points of failure: the server, the client, or the network itself. The debugging strategy outlined below tries to isolate each individual component to find the one that isn't working.

For example, let's look at a sample mount request as made from an NFS client machine:

```
% mount krypton:/usr/src /krypton.src
```

and try to understand how it works and how it can fail. The example asks the server machine 'krypton' to return a file handle (`fhandle`) for the directory `/usr/src`. This `fhandle` is then passed to the kernel in the `nfsmount`(2) system call. The kernel looks up the directory `/krypton.src` and, if everything is okay, it ties the `fhandle` to the directory in a mount record. From now on all file system requests to that directory and below will go through the `fhandle` to the server 'krypton.'

That's how it should work, but if you are reading this it probably didn't. So, how did it fail. First, some general pointers and then we will list the possible errors, and what might have caused them.

General Hints

When there are network or server problems, programs that access hard mounted remote files will fail differently than those which access soft mounted remote files. Hard mounted remote file systems cause programs to retry until the server responds again. Soft mounted remote file systems return an error after trying for a while. `mount` is like any other program, if the server for a remote file system fails to respond it will retry the mount request until it succeeds. A soft mount will try once in the foreground then background itself and keep trying.

Once a hard mount succeeds, programs that access hard mounted files will hang as long as the server fails to respond. In this case, NFS should print a `server not responding` message on the console. On a soft mounted file system programs will get `Connection timed out` (ETIMEDOUT) when they access a file whose server is dead. Unfortunately, many programs in UNIX do

not check return conditions on file system operations so you may not see this error message when accessing soft mounted files. Nevertheless, an NFS error message should be printed on the console in this case also.

If a client is having NFS trouble, check first to make sure the server is up and running. From a client you can type

```
% /usr/etc/rpcinfo -p server_name
```

to see if the server is up at all. It should print out a list of program, version, protocol, and port numbers that resembles:

```
   program vers proto   port
    100004    2   udp   1027  ypserv
    100004    2   tcp   1024  ypserv
    100004    1   udp   1027  ypserv
    100004    1   tcp   1024  ypserv
    100007    2   tcp   1025  ypbind
    100007    2   udp   1035  ypbind
    100007    1   tcp   1025  ypbind
    100007    1   udp   1035  ypbind
    100003    2   udp   2049  nfs
    100016    1   tcp   1032
    100012    1   udp   1087  sprayd
    100011    1   udp   1089  rquotad
    100005    1   udp   1091  mountd
    100008    1   udp   1093  walld
    100002    1   udp   1095  rusersd
    100002    2   udp   1095  rusersd
    100001    1   udp   1098  rstatd
    100001    2   udp   1098  rstatd
    100001    3   udp   1098  rstatd
```

If that works you can also use `rpcinfo` to check if the `mountd` server is running:

```
% /usr/etc/rpcinfo -u server_name 100005 1
```

This should come back with the response:

```
proc 100005 vers 1 ready and waiting
```

If these fail you should go login to the server's console and see if it is okay.

If the server is alive but your machine can't reach it you should check the Ethernet connections between your machine and the server (see *Ethernet Troubleshooting* in the chapter on *Communications* in this manual).

If the server is okay and the network is okay use `ps` to check your client daemons. You should have a `portmap`, `ypbind`, and several `biod` daemons

running. For example, running `ps` sould result in output something like this:

```
% ps ax
32 ?  I      1:07 /etc/portmap
38 ?  I      0:42 /etc/ypbind
61 ?  S      0:45  (biod)
62 ?  S      0:36  (biod)
63 ?  S      0:30  (biod)
64 ?  S      0:27  (biod)
```

The four sections below deal with the most common types of failure. The first tells what to do if your remote mount fails, the next three talk about servers not responding once you have file systems mounted.

Remote Mount Failed

This section deals with problems related to mounting. If `mount` fails for any reason check the sections below for specific details about what to do. They are arranged according to where they occur in the mounting sequence and are labeled with the error message you are likely to see.

`mount` can get its parameters either from the command line or from the file `/etc/fstab` (see `mount`(8)). The example below assumes command line arguments, but the same debugging techniques work if `/etc/fstab` is used in the `mount -a` command.

Keep in mind the interaction of the various players in the mount request. If you understand this, the problem descriptions below will make a lot more sense.

Let's look again at the example `mount` request given above,

```
% mount krypton:/usr/src /krypton.src
```

and see what steps `mount` goes through to mount a remote file system.

1) `mount` opens `/etc/mtab` and checks that this mount has not already been done.

2) `mount` parses the first argument into host 'krypton' and remote directory '/usr/src'.

3) `mount` calls the yellow pages binder daemon `ypbind` to determine which server machine to find the yellow pages server on. It then calls the `ypserv` daemon on that machine to get the internet protocol (IP) address of krypton.

4) `mount` calls krypton's portmapper to get the port number of `mountd`.

5) `mount` calls krypton's `mountd` and passes it `/usr/src`.

6) Krypton's `mountd` reads `/etc/exports` and looks for the exported file system that contains '/usr/src'.

7) Krypton's `mountd` calls the yellow pages server `ypserv` to expand the host names and netgroups in the export list for '/usr/src'.

8) Krypton's `mountd` does a `getfh`(2) system call on '/usr/src' to get the `fhandle`.

9) Krypton's `mountd` returns the `fhandle`.

10) `mount` does an `nfsmount`(2) system call with the `fhandle` and '/krypton.src'.

11) `nfsmount` checks if the caller is superuser and if '/krypton.src' is a directory.

12) `nfsmount` does a `statfs`(2) call to krypton's NFS server (`nfsd`).

13) `mount` opens `/etc/mtab` and adds an entry to the end.

Any one of these steps can fail, some of them in more than one way. The sections below give detailed descriptions of the failures associated with specific error messages.

`/etc/mtab: No such file or directory`

The mounted file system table is kept in the file `/etc/mtab`(5). This file must exist before mount can succeed.

`mount: ... already mounted`

The file system that you are trying to mount is already mounted or there is a bogus entry for it in `/etc/mtab`.

`mount: ... Block device required`

You probably left off the 'krypton:' part of

```
# mount krypton:/usr/src /krypton.src
```

The `mount` command assumes you are doing a local mount unless it sees a colon in the file system name or the file system type is 'nfs' in `/etc/fstab`.

`mount: ... not found in /etc/fstab`

If `mount` is called with only a directory or file system name but not both it looks in `/etc/fstab` for an entry whose file system or directory field matched the argument. For example,

```
# mount /krypton.src
```

will search `/etc/fstab` for a line that has a directory name field of '/krypton.src'. If it finds an entry, such as:

```
krypton:/usr/src /krypton.src nfs rw,hard 0 0
```

it will do the mount as if you had typed

```
# mount krypton:/usr/src /krypton.src
```

If you see this message it means the argument you gave `mount` was not in any of the entries in `/etc/fstab`.

`/etc/fstab: No such file or directory`

`mount` tried to look up the name in `/etc/fstab` but there was no `/etc/fstab`.

`... not in hosts database`

This means the yellow pages could not find the hostname you gave it in the `/etc/hosts` database or that the yellow pages daemon (`ypbind`) is dead on your machine. First check the spelling and the placement of the colon in your mount call. If it looks okay make sure that `ypbind` is running by typing:

```
# ps ax
```

Try `rlogin` or `rcp` to some other machine. If this also fails your `ypbind` is probably dead or hung. If you only get this message for one host name it means that the `/etc/hosts` entry on the yellow pages server needs to be checked. See the section on debugging the yellow pages below in this chapter.

`mount: directory path must begin with '/'`

The second argument to mount is the path of the directory to be covered. This must be an absolute path starting at '**/**'.

`mount: ... server not responding: RPC_PMAP_FAILURE - RPC_TIMED_OUT`

Either the server you are trying to mount from is down, or its portmapper is dead or hung. Try logging in to that machine. If you can log in, try running:

```
# rpcinfo -p hostname
```

You should get a list of registered program numbers. If you don't, you should restart the portmapper. Note that restarting the portmapper requires that you then kill and restart both `inetd` and `ypbind`. To do this, become root and do a

```
# ps ax
```

to find the process ids of `portmap`, `ypbind`, and `inetd`. Next, do a

```
# kill -9 portmap_pid ypbind_pid inetd_pid
```

to kill the daemons. Then type

```
# /etc/portmap
# /etc/inetd
# /etc/ypbind
```

to start new ones.

If you don't want to mess around with this, you can just reboot the server.

If you can't `rlogin` to the server but the server is up, you should check your Ethernet connection by trying `rlogin` to some other machine. You should also check the server's Ethernet connection.

`mount: ... server not responding: RPC_PROG_NOT_REGISTERED`

This means that mount got through to the portmapper but the NFS mount daemon (`rpc.mountd`) was not registered. Go to the server and be sure that `/usr/etc/rpc.mountd` exists and that there is an entry in `/etc/servers` exactly like:

```
rpc udp /usr/etc/rpc.mountd 100005 1
```

Finally, use `ps` to be sure that the internet daemon (`inetd`) is running. If you had to change `/etc/servers` you will have to kill `inetd` and restart it. Look in `/etc/rc.local` to see how it is started at boot time and do that by hand.

`mount: ...: No such file or directory`

Either the remote directory or the local directory doesn't exist. Check spelling. Try to `ls` both directories.

`mount: not in export list for ...`

Your machine name is not in the export list for the file system you want to mount from the server. You can get a list of the server's exported file systems by running

```
# showmount -e hostname
```

If the file system you want is not in the list, or your machine name or netgroup name is not in the user list for the file system, login to the server and check the `/etc/exports` file for the correct file system entry. A file system name that appears in the `/etc/exports` file but not in the output from `showmount`, indicates a failure in `mountd`. Either it could not parse that line in the file, or it could not find the file system, or the filesystem name was not a local mounted file system. See `exports`(5) for more information. If `exports` seems okay check the server's `ypbind` daemon, it may be dead or hung.

`mount: ...: Permission denied`

This message is sort of a generic indication that some authentication failed on the server. It could simply be that you are not in the export list (see above) or the server couldn't figure out who you are (`ypbind` dead), or it could be that the server doesn't believe you are who you say you are. Check the server's `/etc/exports` and `ypbind`. In this case you can just change your hostname with `hostname`(1) and retry the `mount`.

`mount: ...: Not a directory`

Either the remote path or the local path is not a directory. Check spelling and try to `ls` both directories.

`mount: ...: Not owner`

You have to do the mount as root on your machine because it affects the file system for the whole machine, not just you.

Programs Hung

If programs hang doing file related work, your NFS server may be dead. You may see the message `NFS server 192.9.1.186 not responding, still trying` on your console. The message includes the internet address of the NFS server that is down (of course this varies). This is probably a problem either with one of your NFS servers or with the Ethernet. You can figure out which NFS server it is by looking in `/etc/hosts`, or by typing:

```
% ypcat hosts.byname | grep internet_address_#
```

Programs can also hang if an `nd` server dies (see the section on `nd` above) or if a `yp` server dies (see `yp` below).

If your machine hangs completely, check the server(s) from which you have mounted. If one of them (or more) is down don't worry, when the server comes back up your programs will continue automatically and they won't even know the server died. No files will be destroyed.

If a soft mounted server dies, other work should not be affected. Programs that time-out trying to access soft mounted remote files will fail with `errno ETIMEDOUT`, but you should still be able to get work done on your other file systems.

If all of the servers are running go ask someone else using the server or servers that you are using if they are having trouble. If more than one machine is having problems getting service, then it is probably a problem with the server's NFS daemon (`nfsd`(4)). Log in to the server and do a `ps` to see if `nfsd` is running and accumulating cpu time. If not you may be able to kill and then restart `nfsd`. If this doesn't work you will have to reboot the server.

If other people seem to be okay you should check your Ethernet connection and the connection of the server.

Hangs Part Way Through Boot

If your machine comes part way up after a boot, but hangs where it would normally be doing remote mounts, probably one or more servers is down or your network connection is bad. See *Program Hung* and *Remote Mount Failed* above.

Everything Works But Slowly

If access to remote files seems unusually slow, type:

```
# ps aux
```

on the server to be sure it is not being clobbered by a runaway daemon, bad `tty` line, etc. If the server seems okay and other people are getting good response, make sure your block I/O daemons are running; type `ps ax` and look for `biod`. If they are not running or are hung you can kill them off by typing

```
# ps ax | grep biod
```

to find the process ids, and

```
# kill -9 pid1 pid2 pid3 pid4
```

Restart them with

```
# /etc/biod 4
```

To determine whether they are hung, do a `ps` as above, then copy a large remote file, then do another `ps`. If the `biod`s don't accumulate cpu time they are probably hung.

If `biod` is okay check your Ethernet connection. The command `netstat -i` will tell you if you are dropping packets. Also, `nfsstat -c` and `nfsstat -s` can be used to tell if the client or server is doing lots of retransmitting. A retransmission rate of 5% is considered high. Excessive retransmission usually indicates a bad Ethernet board, a bad Ethernet tap, a mismatch between board and tap, or a mismatch between your Ethernet board and the server's board. (See *Ethernet Troubleshooting* in the *Communications* chapter of this manual).

Incompatibilities with Earlier Versions

A few things work differently, or don't work at all, on remote NFS file systems. Here we discuss the incompatibilities and suggest how to work around them.

No Superuser Access Over The Network

Under the NFS a server exports file systems it owns so clients may remote mount them. When a client becomes superuser, it is denied permission on remote mounted file systems. Let's look at the following example:

```
% cd
% touch test1 test2
% chmod 777 test1
% chmod 700 test2
% ls -l test*
-rwxrwxrwx  1 jsbach       0 Mar 24 16:12 test1
-rwx------  1 jsbach       0 Mar 24 16:12 test2
```

Now, retry it again as root.

```
% su
Password:
# touch test1
# touch test2
touch: test2: Permission denied
# ls -l test*
-rwxrwxrwx  1 jsbach       0 Mar 21 16:16 test1
-rwx------  1 jsbach       0 Mar 21 16:12 test2
```

The problem usually shows up during the execution of a set-uid root program. Programs that run as root cannot access files or directories unless the permission for 'other' allows it.

There's another wrinkle to the problem. You cannot change ownership of remote mounted files. Since users cannot do a `chown` command and root is treated as a normal user on remote access, no one but root on the server can change the ownership of remote files. For example, as yourself, you attempt to `chown` a new program, `a.out`, which must be set-uid root. It will not work, as demonstrated here:

```
% chmod 4777 a.out
% su
Password:
# chown root a.out
a.out: Not owner
```

To change the ownership, you must login to the server as root, then make the change. Or, you can move the file to a file system owned by your machine (for example `/usr/tmp` is always owned by the local machine) and make the change there.

(NOTE: Sun does not recommend over-the-net root access as discussed in the following paragraphs.) In a very friendly network environment, you may choose to allow root access over the network. To allow this access, you must change the value of the kernel variable 'nobody' in the NFS server's kernel. To make the change you `adb`(1) the server's kernel. (NOTE: You never change the client's kernel in this procedure, only the server's.)

You can alter the copy of the kernel in memory, in which case the change will be immediate but will be lost when you reboot the machine, or you can change `/vmunix` so that every time you reboot the altered kernel will be run. If you alter `/vmunix` but do not alter the kernel in memory, the machine must be rebooted for the change to take effect. Here are the steps for each kind of change.

- To change the kernel in memory on a running system do the following. (This change will disappear when you reboot, unless you also change the binary image as explained below.) On the NFS server change the value of the kernel variable 'nobody'.

```
# adb -w /vmunix /dev/kmem
```

`adb` responds:

```
not core file = /dev/kmem
```

This is normal. Then you type:

```
nobody/D
```

and `adb` responds with:

```
_nobody:
_nobody:        -2
```

If it does not, stop and call Sun Tech Support for further help. If it does, then you enter:

```
nobody/W0
```

and adb responds with:

```
_nobody:        -2              =       0
```

(Note: the last character in each of the two previous lines is zero.) Then you type <ctrl-D> to exit adb. The kernel currently running will now allow root access to NFS clients.

- To make the change permanent, do the following on a binary image, typically /vmunix:

```
# adb -w /vmunix
```

adb makes no response, so you type:

```
nobody?D
```

and adb responds:

```
_nobody:
_nobody:        -2
```

Again, if adb does not say this, stop and call Sun Tech Support for further help. You then type:

```
nobody?W0
```

and adb responds:

```
_nobody:        -2              =       0
```

(Note: the last character in each of the two previous lines is zero.) Then you type <ctrl-D> to exit adb. From now on, every time you boot /vmunix, the system will allow root access across the NFS.

- If you want to make new kernels that allow root access, you can either go through the procedure above for a binary image each time, or you can go the directory where the kernel object files are and type:

```
# adb -w nfs_server.o
```

You continue as above for in the binary image example. Then every kernel you make with these object files will allow root access on the NFS.

File Operations Not Supported

File locking is not supported on remote file systems. Therefore, the `flock`(2) call will fail when locking a remote file.

In addition append mode and atomic writes are not guaranteed to work on remote files accessed by more than one client simultaneously.

Cannot Access Remote Devices

In the NFS you cannot access a remote mounted device or any other character or block special file — like named pipes.

Beefing Up NFS Security

The NFS currently operates under the assumption that you have a "friendly" net. That is, you can trust all of the users attached to your net. We realize that this does not apply to everyone, so here are some things you can do to improve file security.

Port Monitoring

In Berkeley UNIX, there are some Internet domain source ports to which only privileged users can attach (these are know as "privileged ports"). Currently, the NFS does not check to see if a client is bound to one of these. That is, an NFS server has no way of knowing whether a client's file request originated from the real client's kernel or from some miscreant's user-program. NFS server port checking can be turned on as follows:

```
# adb -w /vmunix
nfs_portmon?W1
_nfs_portmon:    0x0          =        0x1
<ctrl-D>
# adb -w /usr/etc/rpc.mountd
nfs_portmon?W1
_nfs_portmon:    0x0          =        0x1
<ctrl-D>
```

The next time you boot your system, the source ports will be checked. If you can trust all of the root users on your net, then just doing the above is enough. But be warned: some non-UNIX systems do not enforce the privileged port convention (in particular, PCs with 3Com boards). Another warning: all of a server's clients should be running software release 3.0 or higher for this to work (a server will reject all pre-3.0 requests).

IP Source Address Checking

If you are in a situation where you cannot trust all of the root users on your net, then there is more you can do. An NFS server has a list of machine names to which it exports file-systems. This list is know as the export list, and is defined in the file `/etc/exports`. When a client makes a mount request to an NFS server, the server checks to see if the client's machine name appears in its export list.

This checking is not very secure since one can use the `hostname` command to fake a machine-name to something in the server's export list. To improve security, you can have the server also check the client's source IP address. IP addresses can be faked by altering `/etc/hosts`, but the machine being faked will be warned repeatedly. The message "duplicate IP address! sent from ethernet address: XXXXXX" will appear on ts console, displaying the faker's

Ethernet address. To turn on source IP address checking, here's what you do:

```
# adb -w /usr/etc/rpc.mountd
ipaddr_check?W1
_ipaddr_check:  0x0             =       0x1
<ctrl-D>
```

The next time the `rpc.mountd` process runs, it will check the source IP address. You may kill the current `rpc.mountd` process (if any) in order for the changes to take effect immediately, or you can wait until you reboot your system.

Warning: this may not work for clients that are gateways, since they have more than one IP address.

Clock Skew In User Programs

Since the NFS architecture differs is some minor ways from earlier versions of UNIX, you should be mindful of those places where your own programs could run up against these incompatibilities. Be sure to read the section above, *Architectural Incompatibilities*, for a discussion of what will not work over the network.

Because each workstation keeps its own time, the clocks will be out of sync between the NFS server and client. Obviously this might introduce a problem in certain situations. We have fixed all the clock skew problems we have seen. Here are examples of two problems and how they were fixed.

1) Many programs make the (reasonable) assumption that an existing file could not have been created in the future. For example `ls` does it. The command `ls -l` has two basic forms of output, depending upon how old the file is:

```
# date
Jan 22 15:27:01 PST 1985
# touch file2
# ls -l file*
-rw-r--r--  1 root          0 Dec 27  1983 file
-rw-r--r--  1 root          0 Jan 22 15:27 file2
```

The first form of `ls` prints the year, month, and day of last file modification if the file is more than six months old. The second form prints the month, day, and minute of last file modification if the file is less than six months old.

`ls` calculates the age of a file by simply subtracting the modification time of the file from the current time. If the results are greater than six months worth of seconds, the file is 'old.'

Now assume that the time on the server is Jan 22 15:30:31 (three minutes ahead of our local machine's time):

```
# date
Jan 22 15:27:31 PST 1985
# touch file3
# ls -l file*
-rw-r--r--  1 root          0 Dec 27  1983 file
-rw-r--r--  1 root          0 Jan 22 15:26 file2
-rw-r--r--  1 root          0 Jan 22  1985 file3
```

The problem is that the difference of the two times is huge:

(now) - (modification_time) =
(now) - (now + 180 seconds) =
-180 seconds = huge unsigned number,
which is greater than six months.

Thus, `ls` believes the new file was created long ago in the past. Sun modified `ls` to deal with files which are created a short time in the future.

2) `ranlib`(1) was also modified to deal with clock skew. `ranlib` timestamps a library when it is produced, and `ld` compares that timestamp with the last modified date of the library. If the last modified date occurred after the timestamp, then `ld` instructs the user to run `ranlib` again, and reload the library.

   If the library lives on a server whose clock is ahead of the client that runs `ranlib`, `ld` will always complain. Sun fixed `ranlib` to set the timestamp to the maximum value of the current time and the library's modify time.

In general remember, if your application depends upon local time and/or the file system timestamps, then it will have to deal with clock skew problems if it uses remote files.

2.4. yp: The Sun Yellow Pages Service

We begin by introducing `yp` related terms. Following that we list the new `yp` manual pages; that section also includes pointers to `yp` documentation beyond this chapter. Next, you will find explanations of the installation and administration of `yp`, a section on how to debug `yp` when problems occur, and finally, a discussion of file access policies and special security issues raised by the `yp` environment.

What Is The Yellow Pages Service?

The yellow pages is Sun's distributed network lookup service:

- `yp` is a distributed system; the database is fully replicated at several sites, each of which runs a server process for the database. These sites are known as `yp` servers. At steady state, it doesn't matter which server process answers a client request; the answer will be the same all over. This allows multiple servers per network, and gives `yp` service a high degree of availability and reliability.
- `yp` is a lookup service. It maintains a set of databases which may be queried. A client may ask for the value associated with a particular key within a database, and may enumerate every key-value pair within a database.

- yp is a network service. It uses a standard set of access procedures to hide the details of where and how data is stored.

The yp Map

The yp system serves information stored in yp 'maps'. Each map contains a set of keys and associated values. For example, in a map called `hosts.byname`, all the host names within a network are the keys, and the internet addresses of these host names are the values. Each yp map has a `mapname` used by programs to access it. Programs must know the format of the data in the map. Many of the current maps are derived from ASCII files traditionally found in `/etc`: `hosts`, `group`, `passwd`, and a few others. The format of the data within the yp map is identical (in most cases) to the format within the ASCII file. Maps are implemented by `dbm`(3) files located in the subdirectories of the directory `/etc/yp` on yp server machines.

The yp Domain

A yp domain is a named set of yp maps. You can determine and set your yp domain with the `domainname`(1) command. Note that yp domains are different from both Internet domains and `sendmail` domains. A yp domain is simply a directory in `/etc/yp` where a yp server holds all of the yp maps. The name of the subdirectory is the name of the domain. For example, maps for the `literature` domain would be in `/etc/yp/literature`.

A domain name is required for retrieving data from a yp database. Each machine on the network belongs to a default domain set at boot time in `/etc/rc.local` with the `domainname`(1) command. A domain name must be set on all machines, both servers and clients. Further, a single name should be used on all machines on a network.

Masters And Slaves

In the yellow pages environment only a few machines have a set of yp databases. The yp service makes the database set available over the network. A yp client machine runs yp processes and requests data from databases on other machines. Two kinds of machines have databases: a yp slave server and a yp master server. The master server updates the databases of the slave servers. Make changes to databases only on the yp master server. The changes will propagate from the master server to the yp slave servers. If yp databases are created or changed on slave server machines instead of master server machines, the yp's update algorithm will get broken. Always do all the database creation and modification on the master server machine.

A server may be a master with regard to one map, and a slave with regard to another. Random assignment of maps to server machines could introduce a great deal of confusion. We strongly urge you to make a single server the master for all the maps created by `ypinit`(8) within a single domain. This document assumes that one server is the master for all maps in the database.

Yellow Pages Overview

The yellow pages can serve up any number of databases. Typically these include some files that used to be found in `/etc`. For example, before Sun release 2.0 programs would read the `/etc/hosts` file to find an Internet address. When a new machine was added to the network, a new entry had to be added to every machine's `/etc/hosts` file. With the yellow pages, programs that want to look at the `/etc/hosts` file now do a remote procedure call (`rpc`) to the

servers to get the information.

Most of the information describing the structure of the yp system and the commands available for that system is contained in manual pages, and is not repeated here. For quick reference, we list the man pages and an abstract of their contents here. For more information, see the *Network Services Guide*.

ypserv(8) — describes the processes which comprise the yp system. These are ypserv, the yp database server daemon, and ypbind, the yp binder daemon. ypserv must run on each yp server machine. ypbind must run on all machines which use yp services, both servers and clients.

ypfiles(8) — describes the database structure of the yp system.

ypinit(8) — many maps must be constructed from files located in /etc, such as /etc/hosts, /etc/passwd and others. The database initialization tool ypinit(8) does all such construction automatically. In addition, it constructs initial versions of maps required by the system but not built from files in /etc; an example is the map 'ypservers'. Use this tool to set up the master yp server and the slave yp servers for the first time. Do not (generally) use it as an administrative tool for running systems.

ypmake(8) — describes the use of /etc/yp/Makefile, which builds several commonly-changed components of the yp's database. These are the maps built from several ASCII files normally found in /etc: passwd, hosts, group, netgroup, networks, protocols, and services.

makedbm(8) — describes a low-level tool for building a dbm file which is a valid yp map. Databases not built from /etc/yp/Makefile may be built or rebuilt using makedbm. makedbm may also be used to 'disassemble' a map so that you can see the key-value pairs which comprise it. The disassembled form may also be modified with standard tools (such as editors, awk, grep, and cat), and is in the form required for input back into makedbm.

ypxfr(8) — moves a yp map from one yp server to another, using the yp itself as the transport medium. It can be run interactively, or periodically from crontab. In addition, ypserv uses ypxfr as its transfer agent when it is asked to transfer a map.

yppush(8) — describes a tool to administer a running yp system. It is run on the master yp server. It requests each of the ypserv processes within a domain to transfer a particular map, waits for a summary response from the transfer agent, and prints out the results for each server.

ypset(8) — tells a ypbind process (the local one, by default) to get yp services for a domain from a named yp server. This is not for casual use.

yppoll(8) — asks any ypserv for the information it holds internally about a single map.

ypcat(1) — dumps out the contents of a yp map. Use it when you don't care which server's version you are seeing. If you need to see a particular server's map, rlogin to that server (or use rsh) and use makedbm.

ypmatch(8) — prints the value for one or more specified keys in a yp map. Again, you have no control over which server's version of the map you are seeing.

ypwhich(8) — tells you which yp server a node is using at the moment for yp services, or which yp server is master of some named map.

Yellow Pages Installation and Administration

Nine installation and administration topics will be covered:

1) setting up a master yp server
2) altering a yp client's database to use yp services
3) setting up a slave yp server
4) setting up a yp client
5) modifying individual yp maps after yp installation
6) propagating a yp map
7) making new yp maps after yp installation
8) adding a new yp server not in the original set of yp servers
9) changing the master server to a different machine

How To Set Up A Master yp Server

To create a new master server, become superuser and change your current directory to /etc/yp. Then run ypinit(8) with the -m switch. The default domain name and the hostname must be set up; the usual case is that domainname has been set up from /etc/rc.local, and hostname has been set up from /etc/rc.boot. You will be asked whether you want the procedure to die at the first non-fatal error (in which case, you can fix the problem and restart ypinit — this is recommended if you haven't done the procedure before), or to continue despite non-fatal errors. In this second case you can try to fix all the problems by hand, or fix some, then restart ypinit. ypinit will prompt you for a list of other hosts which also will be yp servers. (Initially, this will be the set of yp slave servers, but at some future time any of them might become the yp master server.) You need not add any other hosts at this time, but if you know that you will be setting up some more yp servers, add them now. You will save yourself some work later, and there is no runtime penalty for doing it. (However, don't name every host in the net.)

Prior to running ypinit, the following files in /etc should be complete and reflect an up-to-date picture of your system: passwd, hosts, ethers, group, networks, protocols, and services. In addition, if you know how /etc/netgroup is going to be set up, do that prior to ypinit. If you don't know, ypinit will make an empty 'netgroup' map.

For security reasons you may restrict access to the master yp machine to a smaller set of users than that defined by the complete /etc/passwd. To do this, copy the complete file to someplace other than /etc/passwd, and edit out undesired users from the remaining /etc/passwd. For a security-conscious system, this smaller file should not include the yp escape entry discussed in the next section.

To start providing yellow pages services, invoke /etc/ypserv. It will be started up automatically from /etc/rc.local on subsequent reboots.

Altering A yp Client's Files To Use yp Services

Once the decision has been made to serve a database with the yp, it is desirable that all nodes in the net access the yp's version of the information, rather than the potentially out-of-date information in their local files. That policy is enforced by running a ypbind process on the client node (including nodes which may be running yp servers), and by abbreviating or eliminating the files which traditionally implemented the database. The files in question are: /etc/passwd, /etc/hosts, /etc/ethers, /etc/group, /etc/networks, /etc/protocols, /etc/services, /etc/netgroup, /etc/hosts.equiv, and /.rhosts. The treatment of each file is discussed in this section.

- /etc/networks, /etc/protocols, /etc/ethers, /etc/services, and /etc/netgroup need not exist at any yp client node. If you are squeamish about removing them, move them to backup names; for instance on a machine named 'ypclient':

```
ypclient%  cd /etc
ypclient%  mv networks networks-
ypclient%  <The rest of the renames, similarly>
```

- /etc/hosts.equiv is not normally served by the yp. However, you can add escape sequences to activate the yp. This reduces problems with rlogin, or rsh which are sometimes caused by different /etc/hosts.equiv files on the two machines.

  To let anyone log on to a machine, /etc/hosts.equiv may be edited to contain a single line, with only the character '+' (plus) on it. A line with only a '+' means that all further entries will be retrieved from the yp rather than the local file.

  Alternatively, more control may be exercised over logins by using lines of the form:

```
+@trusted_group1
+@trusted_group2
-@distrusted_group
```

Each of the names to the right of the '@' (at) character is assumed to be a net group name, defined in the global netgroup database. The netgroup database is served by the yp.

If none of the escape sequences is used, only the entries in /etc/hosts.equiv are used; the yp is not used.

- /.rhosts is not normally served by the yp, either. Its format is identical to that of /etc/hosts.equiv. However, since this file controls remote root access to the local machine, unrestricted access is not recommended. Make the list of trusted hosts explicit, or use net group names for the same purpose.

- `/etc/hosts` must contain entries for the local host's name, and the local loopback name. These are accessed at boot time when the `yp` service is not yet available. After the system is running, and after the `ypbind` process is up, the `/etc/hosts` file is not accessed at all. An example of the hosts file for `yp` client 'zippy' is:

```
127.1       localhost
192.9.1.87  zippy   # John Q.  Random
```

- `/etc/passwd` should contain entries for root and the primary users of the machine, and an escape entry to force the use of the `yp` service. A few additional entries are recommended: `daemon`, to allow file-transfer utilities to work; `sync`, to run `sync` on a screwed-up machine before rebooting; and `operator`, to let a dump operator login. A sample `yp` client's `/etc/passwd` file looks like:

```
root:wAm0Y4lEnf6:0:10:God:/:/bin/csh
jrandom:uHPlgQ2:1429:10:J Random:/usr2/jrandom:/bin/csh
operator:VyZr6V9:333:20:sys op:/usr2/operator:/bin/csh
daemon:*:1:1::/:
sync::1:1::/:/bin/sync
+::0:0:::
```

The last line informs the library routines to use the `yp` service rather than give up the search. Entries that exist in `/etc/passwd` mask analogous entries in the `yp` maps. In addition, earlier entries in the file will mask later ones with the same user name, or the same uid. Therefore, please note the order of the entries for daemon and for sync (which have the same uid) and duplicate it in your own file.

By the way, the `/etc/passwd` file on YP servers should not look like this, because the last line (starting with plus) would cause serious security problems.

- `/etc/group` may be reduced to a single line:

```
+:
```

which will force all translation of group names and group ids to be made via the `yp` service. This is the recommended procedure.

How To Set Up A Slave yp Server

The network must be working to set up a slave `yp` server — in particular, you must be able to `rcp` files from the master `yp` server to `yp` slaves. If the master server is a 3.0 release machine, `ypserv`(8) should be running too.

To create a new slave server, change directory to `/etc/yp`. From there run `ypinit`(8) with the **-s** switch, and name a host already set up as a `yp` server, as the master. Ideally, the named host really is the master server, but it can be any host which has its `yp` database set up. The host must be reachable. You must be superuser when you run `ypinit`. The default domain name on the machine

intended to be the yp slave server must be set up, and must be set to the same domain name as the default domain name on the machine named as the master. In addition, an entry for 'daemon' must exist in the /etc/passwd files of both slave and master, and that entry must precede any other entries which have the same uid. (setup creates /etc/passwd; make sure your password file has not been altered to change the order. Note the example shown in the section above.) You won't be prompted for a list of other servers, but you will have the opportunity to choose whether or not the procedure gives up at the first non-fatal error.

After running ypinit, make copies of /etc/passwd, /etc/hosts, /etc/group, /etc/networks, /etc/protocols, /etc/netgroup, and /etc/services. For instance on a machine named 'ypslave':

```
ypslave% cp /etc/passwd /etc/passwd-
```

Edit the original files in accordance with the section above on altering the client's database, so as to insure that processes on the slave yp server will actually make use of the yp services, rather than the local ASCII files. (That is, make sure the yp slave server is also a yp client) Make backup copies of the edited files, as well. For instance:

```
ypslave% cp /etc/passwd /etc/passwd+
```

After the yp database gets set up by ypinit, type /etc/ypserv to begin supplying yp services. On subsequent reboots, it will start automatically from /etc/rc.local

How To Set Up A yp Client

To set up a yp client, edit the local files as described in the section above on altering a yp client's file database. If /etc/ypbind is not running already, start it. With the ASCII databases of /etc abbreviated and /etc/ypbind running, the processes on the machine will be clients of the yp services. At this point, there must be a yp server available; all sorts of stuff will hang up if no yp server is available while ypbind is running. Note the possible alterations to the client's /etc database as discussed above in the section on altering the client. Because some files may not be there, or some may be specially altered, it is not always obvious how the ASCII databases are being used. The escape conventions used within those files to force inclusion and exclusion of data from the yp databases are found in the following man pages: passwd(5), hosts(5), netgroup(5), host.equiv(5), group(5). In particular, notice that changing passwords in /etc/passwd (by editing the file, or by running passwd(1)), will only affect the local client's environment. Change the yp password database by running yppasswd(1).

How To Modify Existing yp Maps After yp Installation

Databases served by the yp must be changed ON THE MASTER SERVER. The databases expected to change most frequently, like /etc/passwd, may be changed by first editing the ASCII file, and then running make(1) on /etc/yp/Makefile — also see ypmake(8).

Non-standard databases (that is, databases which are specific to the applications of a particular vendor or site, but which are not part of Sun's release), or databases which are expected to change rarely, or databases for which no ASCII form exists (for example, databases not around before the yp), may be modified "manually". The general procedure is to use makedbm(8) with the **-u** switch to disassemble them into a form which can be modified using standard tools (such as awk, sed, or vi). Then build a new version again using makedbm(8). This may be done by hand in two ways:

1) The output of makedbm can be redirected to a temporary file which can be modified, then fed back into makedbm, or

2) The output of makedbm can be operated on within a pipeline which feeds into makedbm again directly. This is appropriate if the disassembled map can be updated by modifying it with awk, sed, or a cat append, for instance.

Suppose we want to create a non-standard yp map, called 'mymap'. We want it to consist of key-value pairs in which the keys are strings like al, bl, cl, etc. (the "l" is for "left"), and the values are ar, br, cr (the "r" is for "right"). There are two possible procedures to follow when creating new maps. In one we use an existing ASCII file as input; in the other we use standard input.

For example, suppose there is an existing ASCII file named '/etc/yp/mymap.asc,' created with an editor or a shell script on our machine 'ypmaster'. 'home_domain' is the subdirectory where the map is located. We can create the yp map for this file by:

```
ypmaster%  cd /etc/yp
ypmaster%  makedbm mymap.asc home_domain/mymap
```

But at this point we notice the map really should have included another 2-tuple: (dl, dr). We can make the modification quite simply. In all situations like this, remember to modify the map by first modifying the ASCII file. Modifications made to the map, not also in the ASCII file, will be lost. Make the modification like this:

```
ypmaster%  cd /etc/yp <if not already there>
ypmaster%  <make editorial change to mymap.asc>
ypmaster% makedbm mymap.asc home_domain/mymap
```

When there is no original ASCII file, we can create the yp map from the keyboard by typing input like this (our machine name is 'ypmaster', and the default domain is 'home_domain'):

```
ypmaster% cd /etc/yp
ypmaster% makedbm - home_domain/mymap
al ar
bl br
cl cr
<ctl D>
```

When you need to modify that map, you can use makedbm to create a temporary ASCII intermediate file which can be edited using standard tools. For instance:

```
ypmaster%  cd /etc/yp
ypmaster%  makedbm -u home_domain/mymap > mymap.temp
```

At this point 'mymap.temp' can be edited to contain the correct information. A new version of the database is created by the commands

```
ypmaster%  makedbm mymap.temp home_domain/mymap
ypmaster%  rm mymap.temp
```

The preceding paragraphs explained how to use some tools, but in reality almost everything you actually have to do can be done by ypinit(8) and /etc/yp/Makefile, unless you add non-standard maps to the database, or change the set of yp servers after the system is already up and running.

Whether you use the Makefile in /etc/yp or some other procedure — Makefile is one of many possible — the goal is the same: a new pair of well-formed dbm files must end up in the domain directory on the master yp server.

Propagation Of A yp Map

To propagate a map means to move it from place to place — in general, to move it from the master yp server to a slave yp server. Initially, ypinit(8) moves it, as described above in the section *How To Set Up A Slave* yp *Server*. After a slave yp server has been initialized, updated maps are transferred from the master server by ypxfr(8). ypxfr may be run in three different ways: periodically by cron(8); by ypserv(8); and interactively by a user. Let's look an example of each.

Maps have differing rates of change; for instance 'protocols.byname' may not change for months at a time, but 'passwd.byname' may change several times a day in a large organization. You can set up crontab(5) entries to periodically run ypxfr at a rate appropriate for any map in your yp database. ypxfr will contact the master server and transfer the map only if the master's copy is more recent than the local copy.

To avoid a crontab entry for each map, several maps with approximately the same change characteristics can be grouped in a shell script, and the shell script can be run from /usr/lib/crontab. Suggested groupings, mnemonically named, can be found in /etc/yp: ypxfr_1perhour, ypxfr_1perday, and ypxfr_2perday. If the rates of change are inappropriate for your environment, these shell scripts can be easily modified or replaced.

These same shell scripts should be run at each yp slave server in the domain. Alter the exact time of execution from one server to another, so as to prevent the checking from bogging down the master. If you want the map transferred from some particular server, not the master, that can be specified (using ypxfr's '-h' option) within the shell script. Finally, maps having unique change characteristics can be checked and transferred by explicit invocations of ypxfr within crontab.

ypxfr also gets invoked by ypserv, responding to a 'Transfer Map' request. That request is made as an RPC call from yppush(8). yppush is run on the master yp server. It enumerates the yp map 'ypserver' to generate a list of yp servers in your domain. To each of the named yp servers, it sends a 'Transfer Map' request. ypserv forks off a copy of ypxfr, invoking it with the '-C' flag, and passing it the information it needs to identify the map, and to call back the initiating yppush process with a summary status.

Release 3.0 ypserv also kicks off ypxfr when it attempts to emulate the behavior of ypserv from an earlier release, for instance when a 2.x master ypserv communicates directly with the 3.0 ypserv.

In the cases mentioned above, ypxfr's transfer attempts and the results can be captured in a log file. If /etc/yp/ypxfr.log exists, results will be appended to it. No attempt to limit the log file is made - you are in charge of that. To turn off logging, remove the log file.

In the third case the user runs ypxfr as a command. Typically, one does this only in exceptional situations — for example when setting up a temporary yp server to create a test environment, or when trying to quickly get a yp server which has been out of service consistent with the other servers.

How To Make New yp Maps After yp Installation

Adding a new yp map entails getting copies of the map's dbm files into the domain directory on each of the yp servers in the domain. The actual mechanism has been described above in *Propagation Of A yp Map*. This section will only describe the work required to get the proper mechanisms in place so the propagation works correctly. Things must be set up correctly on both the master and the slaves.

After deciding which yp server is the master of the map, you should modify /etc/yp/Makefile on the master server so that the map can be conveniently rebuilt. Actual, case by case modification is too varied to describe here, but typically a human-readable ASCII file is filtered through awk, sed, and/or grep to make it suitable for input to makedbm(8). Consult the existing Makefile as a source for programming examples. We urge you make use of the mechanisms already in place in /etc/yp/Makefile when deciding how to create dependencies that make(1) will recognize; specifically, the use of .time files allows you to see when the Makefile was last run for the map.

Support on the yp slave servers for propagation of the new maps consists of appropriate entries either in /usr/lib/crontab, or in one of the ypxfr shell scripts mentioned in the previous section. To get an initial copy of the map, ypxfr can be run by hand on each of the slave servers. The map must be globally available before clients begin to access it. If the map is available from

some yp servers, but not all, you will see unpredictable behavior from client programs.

How To Add A New yp Server Not In The Original Set

To add a new yp slave server start by modifying some maps on the master yp server. If the new server is a host which has not been a yp server before, the host's name must be added to the map 'ypservers' in the default domain. The sequence for adding a server named 'ypslave' to domain 'home_domain' is:

```
ypmaster% cd /etc/yp
ypmaster% (makedbm -u home_domain/ypservers;\
echo ypslave ypslave)|makedbm - tmpmap
ypmaster% mv tmpmap.dir home_domain/ypservers.dir
ypmaster% mv tmpmap.pag home_domain/ypservers.pag
ypmaster% yppush ypservers
```

Note that we display some commands above on two lines. You may type these as one long command (even if the line wraps on your screen), or you may escape the return and newline with a backslash, as shown here. However, you cannot simply type in half the command, hit return, and type the second half.

The host's address should be in 'hosts.byname'. If that's not true, edit `/etc/hosts` and run `make`. In this case the commands should be:

```
ypmaster%  <edit /etc/hosts here>
ypmaster% cd /etc/yp
ypmaster% make hosts
```

The new slave yp server's databases should be set up by copying the databases from yp master server 'ypmaster'. Remote login to the new yp slave, and use `ypinit`(8) in the following way:

```
ypslave% cd /etc/yp
ypslave% ypinit -s ypmaster
```

Then complete the steps described above in the section *How To Set Up A Slave yp Server.*

How To Change The Master Server

To change a map's master, first build the map at the new master. Because the old yp master's name occurs as a key-value pair in the existing map, it is not sufficient to use an existing copy at the new master, or to send a copy there with `ypxfr`. The key must be reassociated with the new master's name. If the map has an ASCII source file, it should be present in its current version at the new master. Remake the yp map (we'll call it 'jokes.bypunchline') locally with the sequence:

```
newmaster% cd /etc/yp
newmaster% make jokes.bypunchline
```

/etc/yp/Makefile must be set up correctly for this to work. If it isn't, do it now. In addition, this is a good time to go back to the old master (if it will remain a yp server) and edit /etc/yp/Makefile so that 'jokes.bypunchline' is no longer made there — that is, comment out the section of oldmaster:/etc/yp/Makefile which made 'jokes.bypunchline'.

If the map only exists as a dbm data base, you can remake it on the new master by disassembling an existing copy (one from any yp server will do) and running the disassembled version back through makedbm. For example:

```
newmaster% cd /etc/yp
newmaster% ypcat -k jokes.bypunchline |\
makedbm - mydomain/jokes.bypunchline
```

After making the map on the new master, you must send a new copy of the map to the other (slave) yp servers. However, don't use yppush — the other slaves will try to get new copies from the old master, rather than the new one.

A typical method (you may find others) is to become superuser on the old master server and type:

```
oldmaster% /etc/yp/ypxfr -h newmaster jokes.bypunchline
```

Now you have a new copy on the old master, and now you may run yppush. The remaining slave servers still believe that the old master is the current master, and will attempt to get the current version of the map from the old master. When they do so, they will get the new map, which names the new master as the current master.

If the method above fails, there is a cumbersome but sure-fire option. On each yp server machine, while superuser, execute the command shown just above. This will certainly work, but should be considered the worst case solution.

Debugging A Yellow Pages Client

We divide this debugging section into two parts — first those problems seen on a yp client, and then those problems seen on a yp server.

Before trying to debug a yellow pages client, read the earlier section in this chapter on how the yp works.

On Client: Commands Hang

The most common problem at a yp client node is for a command to hang and generate console messages which say:

```
yp: server not responding for domain <wigwam>.  Still trying
```

Sometimes many commands will begin to hang, even though the system as a whole seems okay and you can run new commands.

The message above indicates that ypbind on the local machine is unable to communicate with ypserv in the domain 'wigwam'. This often happens when machines which run ypserv have crashed. It may also occur if the net or the

yp server machine is so overloaded that ypserv can't get a response back to your ypbind within the timeout period. Under these circumstances, all the other yp client nodes on your net will show the same or similar problems. The condition is temporary in most cases, and the messages will usually go away when the yp server machine reboots and ypserv gets back in business; or when the load on the yp server nodes and/or the Ethernet decreases.

However, in the circumstances described below, the situation will never get better.

- The yp client has not set, or has incorrectly set, domainname on the machine. Clients must use a domain name that the yp servers know. Use domainname(1) to see the client domain name. Compare that with the domain name set on the yp servers. The domain name should be set in /etc/rc.local. When /etc/rc.local fails to set, or incorrectly sets, domainname you will have to do the following: become superuser on the machine in question, edit /etc/rc.local to fix the domainname line with a proper domain name (this assures domain name will be correct every time the machine boots), and set domainname 'by hand' so it is fixed immediately — type:

```
# domainname good_domain_name
```

- If your domain name is correct, make sure your local net has at least one yp server machine. You can only bind to a ypserv process on your local net, not on another accessible net. There must be at least one yp server for your machine's domain running on your local net. Two or more yp servers will improve availability and response characteristics for yp services.
- If your local net has a yp server, make sure it is up and running. Check other machines on your local net. If several client machines have problems simultaneously, suspect a server problem. Find a client machine behaving normally, and try the ypwhich command. If ypwhich never returns an answer, kill it and go to a terminal on the yp server machine. Type:

```
% ps ax | grep yp
```

and look for ypserv and ypbind processes. If the server's ypbind daemon is not running, start it up by typing:

```
% /etc/ypbind
```

If there is a ypserv process running, do a ypwhich on the yp server machine. If ypwhich returns no answer, ypserv has probably hung and should be restarted. Kill the existing ypserv process (you must be logged on as root), and start /etc/ypserv:

```
% kill -9 [some pid # from ps]
% /etc/ypserv
```

If ps shows no ypserv process running, start one up.

On Client: yp Service Unavailable

When other machines on the network appear to be okay, but yp service becomes unavailable on your machine, many different symptoms may show up. Among them: some commands appear to operate correctly while others terminate printing an error message about the unavailability of yp; some commands limp along in a backup-strategy-mode particular to the program involved; and some commands or daemons crash with obscure messages or no message at all. For example, things like the following may show up:

```
my_machine% ypcat myfile
ypcat: can't bind to yp server for domain <wigwam>.
    Reason: can't communicate with ypbind.
```

```
my_machine% /etc/yp/yppoll myfile
Sorry, I can't make use of the yellow pages.  I give up.
```

When symptoms like those above occur, try

```
my_machine% ls -l
```

on a directory containing files owned by many users, including users not in the local machine's /etc/passwd file — for example /usr. If the ls -l reports file owners not in the local machine's /etc/passwd file as numbers, rather than names, it is one more symptom that yp service is not working.

These symptoms usually indicate that your ypbind process is not running. You can do a ps ax to check for one. If it you do not find it, type:

```
my_machine% /etc/ypbind
```

to start it. yp problems should disappear.

On Client: ypbind Crashes

If ypbind crashes almost immediately each time it is started, you should look for a problem in some other part of the system. Check for the presence of the portmap daemon by typing:

```
my_machine%  ps ax | grep portmap
```

If you don't find it running, reboot.

If portmap itself will not stay up or behaves strangely, look for more fundamental problems. Check the network software in the ways suggested in the

section on *Ethernet Debugging* in the *Communications* chapter of this manual.

You may be able to talk to the `portmap` on your machine from a machine operating normally. From such a machine, type:

```
flipper% rpcinfo -p your_machine_name
```

If your `portmap` is okay, the output should look like:

```
[program, version, protocol, port]:

[100005, 1, 17, 1046]
[100001, 2, 17, 1055]
[100001, 1, 17, 1055]
[100002, 1, 17, 1052]
[100008, 1, 17, 1049]
[100007, 1, 17, 1027]
[100007, 1, 6, 1026]
[100007, 2, 17, 1031]
[100007, 2, 6, 1030]
```

On your machine the port numbers will be different. The four entries that represent the `ypbind` process are:

```
[100007, 1, 17, port_#]
[100007, 1, 6, port_#]
[100007, 2, 17, port_#]
[100007, 2, 6, port_#]
```

If they are not there, `ypbind` has been unable to register its services. Reboot the machine. If they are there and they change each time you try to restart `/etc/ypbind`, reboot the system, even if the `portmap` is up. If the situation persists after reboot, call for help.

On Client: `ypwhich` Inconsistent

When you use `ypwhich` several times at the same client node, the answer you get back varies — the `yp` server changes. This is normal. The binding of `yp` client to `yp` server will change over time on a busy net, and when the `yp` servers are busy. Whenever possible, the system stabilizes at a point where all clients get acceptable response time from the `yp` servers. As long as your client machine gets `yp` service, it doesn't matter where the service comes from. Often a `yp` server machine gets its own `yp` services from another `yp` server on the net.

Debugging A Yellow Pages Server

Before trying to debug a yellow pages server, read the earlier section in this chapter on how the `yp` works.

On Server: Different Versions Of A yp Map

Since yp works by propagating maps among servers, you will sometimes find different versions of a map at servers on the network. This version skew is normal if transient, and abnormal otherwise.

Most commonly, normal update is prevented when some yp server or some gateway machine between yp servers is down during a map transfer attempt. (Normal update is described in the section above, *Propagation Of A* yp *Map*). When all the yp servers, and all the gateways between them, are up and running, ypxfr should succeed.

If a particular slave server has problems updating, login to that server and run ypxfr interactively. If ypxfr fails, it will tell you why it failed, and you can understand and fix the problem. If ypxfr succeeds, but you believe it fails sometimes, create a log file to enable logging of messages. Type:

```
ypslave% cd /etc/yp
ypslave% touch ypxfr.log
```

This saves all output from ypxfr. The output looks much like what ypxfr creates when run interactively, but each line in the log file is timestamped. (You may see funny orderings in the timestamps. That's OK — the timestamp tells you when ypxfr began its work. If copies of ypxfr ran simultaneously, but their work took differing amounts of time, they may actually write their summary status line to the log files in an order different from the order of invocation.) Any pattern of intermittent failure will show up in the log. When you have fixed the problem, turn off logging by removing the log file. If you forget to remove it, it will grow without limit.

While still logged in to the problem yp slave server, inspect /usr/lib/crontab and the ypxfr* shell scripts it invokes. Typos in these files will cause propagation problems, as will failures to refer to a shell script within crontab, or failures to refer to a map within any shell script.

Also, make sure that the yp slave server is in the map ypservers within the domain. If it's not, it will still work fine as a server, but yppush won't tell it when a new copy of a map exists.

If the problem is not obvious, you can work around it while you debug by using rcp(1) or tftp(1) to copy a recent version from any healthy yp server. You may not be able to do this as root, but you probably will be able to do it as daemon. For instance, to transfer map 'busted':

```
ypslave% chmod go+w /etc/yp/mydomain
ypslave% su daemon
$ rcp ypmaster:/etc/yp/mydomain/busted.\* /etc/yp/mydomain
$ ^D
ypslave% chown root /etc/yp/mydomain/busted.*
ypslave% chmod go-w /etc/yp/mydomain
```

Notice that the '*' character has been escaped in the command line, so that it will be expanded on 'ypmaster', instead of locally on 'ypslave'. In addition, notice

that the map files should be owned by root, so you must change ownership of them after the transfer. Obviously, if you can do the `rcp` as root, it makes the whole thing easier.

On Server: `ypserv` Crashes

When the `ypserv` process crashes almost immediately, and won't stay up even with repeated activations, the debug process is virtually identical to that described above in the section *On Client: ypbind Crashes*. Check for the `portmap` daemon:

```
ypserver% ps ax | grep portmap
```

Reboot the server if you do not find it. If it is there, type:

```
ypserver% /usr/etc/rpcinfo -p
```

and look for output to the screen like:

```
[program, version, protocol, port]:

[100001, 2, 17, 1062]
[100001, 1, 17, 1062]
[100002, 1, 17, 1060]
[100008, 1, 17, 1058]
[100005, 1, 17, 1056]
[100007, 1, 17, 1032]
[100007, 1, 6, 1027]
[100004, 1, 6, 1026]
[100004, 1, 17, 1024]
[100004, 2, 6, 1043]
[100004, 2, 17, 1040]
```

On your machine, the port numbers will be different. The four entries that represent the `ypserv` process are:

```
[100004, 1, 6, port_#]
[100004, 1, 17, port_#]
[100004, 2, 6, port_#]
[100004, 2, 17, port_#]
```

If they are not there, `ypserv` has been unable to register its services. Reboot the machine. If they are there, and they change each time you try to restart `/etc/ypserv`, reboot the machine. If the situation persists after reboot, call for help.

Yellow Pages Policies

Here are the policies set by the C-library routines when they access the following files on a system running the yellow pages.

`/etc/passwd`
Always consulted. If there are + or − entries, the yp password map is consulted, otherwise yp is unused.

`/etc/group`
Always consulted. If there are + or − entries, the yp group map is consulted, otherwise yp is unused.

`/etc/hosts.equiv`
(And similarly for `.rhosts`) Always consulted, though neither of these files is in the yellow pages database. (See the section below *How Security Is Changed With The Yellow Pages*, for a fuller explanation of these two files.) If there are + or − entries, whose arguments are netgroups, the yp netgroup map is consulted, otherwise yp is unused.

`/etc/services`
Never consulted. The data that was formerly read from this file now comes from the yp services database.

`/etc/protocols`
Never consulted. The data that was formerly read from this file now comes from the yp protocols database.

`/etc/networks`
Never consulted. The data that was formerly read from this file now comes from the yp networks database.

`/etc/netgroup`
Never consulted. The data that was formerly read from this file now comes from the yp netgroup database.

`/etc/ethers`
Never consulted. The data read from this file comes from the yp netgroup database.

`/etc/hosts`
Consulted only when booting (by the `ifconfig` command in the `/etc/rc.boot` file). After that the yp is used instead.

How Security Is Changed With The Yellow Pages

Read the section above on yp accessing policies to better understand yp security issues.

Global And Local yp Database Files

Of the yp databases, six were formerly in `/etc`: `/etc/passwd`, `/etc/group`, `/etc/hosts`, `/etc/networks`, `/etc/services`, and `/etc/protocols`. In addition, there are two new files `/etc/netgroup` and `/etc/ethers`. (Note that a site may add database files of its own.) We divide the yellow pages into local and global file types. A local file is first checked for on your own machine, then in the yellow pages. A global file is only checked for in the yellow pages. `/etc/passwd` and `/etc/group` are the local files in the yellow pages database. The other five yellow pages files are

global.

For example, a program that calls `/etc/passwd` (a local file) will first look in the password file on your machine; the yellow pages password file will only be consulted if your machine's password file contains '+' (plus sign) entries. The `/etc/passwd` file is local so that you can control the entries for your own machine. The only other local file is `/etc/group`. To repeat, local files are consulted first on your own machine, before looking in the yellow pages.

The remaining yellow pages files (`hosts`, `networks`, `ethers`, `services`, `protocols`, and `netgroup`) are global files. The information in these files is network wide data, and is accessed only from the yellow pages. However, when booting, each machine needs an entry in `/etc/hosts` for itself. In summary, if yellow pages is running, global files are only checked in the yellow pages; a file on your local machine is not consulted.

Two Other Files yp Consults

The files `/etc/hosts.equiv` and `/.rhosts` are not in the yellow pages database. Each machine has its own unique copy. However it is possible to put entries in your `/etc/hosts.equiv` file that refer to the yellow pages. For example a line consisting of

```
+@engineering
```

will include all members of `engineering` as it is defined in the local file `/etc/netgroup` or in the yp database. A line consisting only of '+' (a plus sign) will include everyone in your `/etc/hosts.equiv` file.

Security Implications

Recall that to be able to log into a machine without having a password, you need to be in both the `/etc/hosts.equiv` file and the `/etc/passwd` file. By having a '+' entry in `/etc/hosts.equiv`, you effectively bypass this check, and anyone in your `/etc/passwd` file will be allowed to rlogin to your machine without restriction.

An `/etc/passwd` file and `/etc/group` file may also have '+' entries. A line in an `/etc/passwd` file such as

```
+nb:::Napoleon Bonaparte:/usr2/nb:/bin/csh
```

pulls in an entry for `nb` from the yellow pages. It gets the `uid`, `gid` and `password` from the yellow pages, and gets the gecos, home directory and default shell from the '+' entry itself. On the other hand, an `/etc/passwd` entry such as

```
+nb:
```

gets all information from the yellow pages. Finally, notice that

```
+nb::1189:10:Napoleon Bonaparte:/usr2/nb:/bin/csh
```

is quite different from

```
nb::1189:10:Napoleon Bonaparte:/usr2/nb:/bin/csh
```

In the first of the two examples the password field is obtained from the yellow pages, in the second user `nb` has no password. Also, if there is no entry for `nb` in the yellow pages, then the effect of the first example is as if no entry for `nb` was present at all.

Warning: do not put the line

```
+::0:0:::
```

in the `yp` server's `/etc/passwd` file! It would render every machine on the network insecure.

Special yp Password Change

When you change your password with the `passwd`(1) command, you will change the entry explicitly given in your own `/etc/passwd` file. If your password is not given explicitly, but rather is pulled in from the yellow pages with a '+' entry, then the `passwd` command will print the error message `Not in passwd file`. To change your passwd in the yellow pages, you must use the new `yppasswd`(1) command. In order to enable this service, the system administrator must start up the daemon `yppasswdd`(8C) server on the machine serving as the master for the yellow pages password file.

Netgroups: Networkwide Groups Of Machines And Users

Netgroups are networkwide groups of machines and users defined in the `/etc/netgroup`(5) file on the master yellow pages server. These groups are used for permission checking during remote mount, login, remote login, and remote shell.

The master `yp` server uses `/etc/netgroup` to generate three `yp` maps in the `/etc/yp/`*domainname* directory: `netgroup`, `netgroup.byuser` and `netgroup.byhost`. The `yp` map `netgroup` contains the basic information in `/etc/netgroup`. The two other `yp` maps contain a more specific form of the information to speed up the lookup of netgroups given the host or user.

The programs that consult the `yp` maps are `login`(1), `mountd`(8), `rlogin`(1) and `rsh`(1). `login` consults them for user classifications if it encounters netgroup names in `/etc/passwd`(5). `mountd` consults them for machine classifications if it encounters netgroup names in `/etc/exports`(8). `rlogin` and `rsh` consult the `netgroup` map for both machine and user classifications if they encounter netgroup names in */etc/hosts.equiv*(5) or *.rhosts*.

Here is a sample `/etc/netgroup` file. See `netgroup`(5) for a description of file format and definition of lines and fields.

```
#
# Engineering: Everyone, but eric, has a machine; he has no machine.
# The machine 'testing' is used by all of hardware.
#
engineering    hardware software
hardware  (mercury,alan,sun) (venus,beth,sun) (testing,-,sun)
software  (earth,chris,sun) (mars,deborah,sun) (-,eric,sun)
#
# Marketing: Time-sharing on jupiter
#
marketing (jupiter,fran,sun) (jupiter,greg,sun) (jupiter,dan,sun)
#
# Others
#
allusers (-,,sun)
allhosts (,-,sun)
```

Based on the above, the users will be classified into groups as follows:

GROUP	USERS
hardware	alan, beth
software	chris, deborah, eric
engineering	alan, beth, chris, deborah, eric
marketing	fran, greg, dan
allusers	(every user in the yp map passwd)
allhosts	(no users)

And here is how the machines would be classified:

GROUP	HOSTS
hardware	mercury, venus, testing
software	earth, mars
engineering	mercury, venus, earth, mars, testing
marketing	jupiter
allusers	(no hosts)
allhosts	(all hosts in the yp map hosts)

Manual Pages Covering Security Issues

For more details, see the following man pages: `yppasswd`(1), `hosts.equiv`(5), `export`(5), `passwd`(5), `group`(5), `netgroup`(5), `yppasswdd`(8C).

What If You Do Not Use The Yellow Pages?

If you choose not to use the yellow pages, the procedure for bypassing the software implementation is quite simple. In the file `/etc/rc.local` find the lines that look like:

```
if [ -f /etc/ypbind ]; then
    /etc/ypbind; echo -n ' ypbind'  >/dev/console
fi
```

And comment them out, like:

```
# if [ -f /etc/ypbind ]; then
#     /etc/ypbind; echo -n ' ypbind'    >/dev/console
# fi
```

2.5. Adding A New User To A Machine

To add a new user, add an entry to the password file and create a home directory on the new user's machine as described in the steps below. To add a new client machine, see *Adding A Client Machine To An* `nd` *Server, below.*

Edit The Master `yp` Server's `/etc/passwd` File

Typically, for a new user, add a password file entry to every machine on the local network. You must be superuser to do it, and you should begin on the master `yp` server machine. First edit the master `yp` server's `/etc/passwd` file. Later the password file entry for the user will be copied to the `/etc/passwd` file on the new client's partition; without an entry in it, the person administering the new client machine would not be able to login should the yellow pages fail.

On the master `yp` server add a new line to the password file with the `vipw` command; `vipw` brings the password file into the `vi` editor, and prevents anyone else from editing it until you are done:

```
# /etc/vipw
```

`/etc/passwd` is a readable ASCII file with a one line entry for each valid user on the system. Each entry is separated into fields by colons (:); there are seven fields on each line and some fields may be left blank by placing two colons back to back. Avoid using the characters for single and double quotes (‘ ’ "), backslashes (\), and parentheses (()) in the password file. See `passwd`(5) for more about the file format.

Let us suppose that your new user's name is Mr. Chimp and his account is going to be 'bonzo', you would add a line similar to the one shown below.

```
bonzo::1947:10:Mr.  Chimp:/usr2/bonzo:/bin/csh
```

Notice that the second field is blank in the example. This field, when filled, contains an encrypted version of the user's password. However, when the field is blank, anyone can login simply by typing the user name — no password is required. You cannot create a password by making an entry in the

/etc/passwd file. You must use passwd(1) while logged in as the user, or as superuser. Since anyone can login when a user has no password, you may provide a password for the new user and let him know it so he can log in and change it to whatever he prefers. When Mr. Chimp logs in for the first time, he can use the passwd utility to change his password, or yppasswd(1) to change it in the yp database.

After Mr. Chimp has a password, the entry for 'bonzo' in the password file will look something like:

```
bonzo:3u0mRdrJ4tEVs:1947:10:Mr.  Chimp:/usr2/bonzo:/bin/csh
```

Fields in the password file have the following meanings:

1) Login name — synonymous with user name.

2) Encrypted password. Tell all new users how to add, or change, their password with the passwd command and the yppasswd command. Remember, the system administrator can make this field empty when a user has forgotten his or her password, thereby enabling login without a password until such time as a new one is given. Note that an asterisk (*) in this field matches no password. The user can only log in if the machine name of the machine he is logging in from is in the /etc/hosts.equiv file on the local machine.

3) User ID. A number unique to this user. A system knows the user by ID number associated with login name, therefore a login name must have the same user ID number on all password files of machines which are networked in a local domain. Failure to keep ID's unique will prevent moving files between directories on different machines because the system will respond as if the directories are owned by two different users. In addition, file ownership may become confused when an NFS server exports a directory to an NFS client whose password file contains users with uid's that match those of different users on the NFS server.

4) Group ID. Can be used to group users together who are working on similar projects. All system staff are in group '10' for historical reasons. In this example Mr. Chimp is in the system staff group. Do not put normal users in this group. If you are not sure what group to put a new client in, see group(5) and look in the file /etc/group.

5) Information about user — usually real name, phone number, etc. An ampersand (&) here is shorthand for the user's login name.

6) The user's home directory — the directory the user logs in to.

7) Initial shell to use on login. If this field is blank the default /bin/sh is used. We recommend placing /bin/csh here, as in the example above, it gives a Berkeley 4.2 C-Shell as the user's initial shell.

After you have updated the password file and created a password for the new user, be sure to update the yellow database by running /etc/yp/make for /etc/passwd:

```
# cd /etc/yp
# make passwd
```

Make A Home Directory

After adding a new entry to the password file, you should create a home directory for the new user to login to. This will be the same as the directory given in the sixth field of the password file entry. In the `/usr2` directory make a directory for the new user and change ownership to the user's login name and change group to the user's group. For example:

```
# cd /usr2
# mkdir bonzo
# chown bonzo bonzo
# chgrp 10 bonzo
```

Note that if the yellow pages databases for the password file have not yet been updated on the machine's yellow pages server, you will get the following error message when you attempt to do the `chown`:

```
unknown user id:  username
```

In that case, you can use the following set of commands:

```
# cd /usr2
# mkdir bonzo
# chown userid# bonzo
# chgrp 10 bonzo
```

You use Mr. Chimp's user id number (from the password file entry) instead of login name to change the ownership of his home directory.

The New User's Environment

Finally, you may define the new user's environment on login in several ways. For example you may give her a copy of such files as `.login` and `.cshrc` if she uses '/bin/csh', or `.profile` if she uses '/bin/sh'. See the `csh`(1) and `sh`(1) pages in the *Commands Reference Manual* for discussion of these files, they can automatically set up the terminal and shell environment at each login. You may also want to give a user `.suntools` and `.mailrc` files.

If the new user is a member of any groups at your site, add her to `/etc/group` as necessary — see `group`(5) and `groups`(1). Be sure to make the changes to the `/etc/group` and `/etc/netgroup` files on the master `yp` server if you run the yellow pages.

If this user should belong to any 'mailing lists,' make the addition to `/usr/lib/aliases`. See the section on *Setting Up The Postmaster Alias* in *Setting Up The Mail Routing System* in chapter 4 of this manual.

2.6. Adding A Client Machine To An nd Server

Each client machine of an nd server machine owns a root and a swap partition on a network disk. A public partition, holding a UNIX kernel, boot program, and other information and programs necessary for booting, is shared between the file server and all the clients. Typically, the nd server also exports file systems to its clients using the NFS.

To add a client, the nd server must have soft partitions available on the network disk allocated for client use. For a further discussion of network disks, including usage and terminology, turn to the earlier section in this chapter, *The Network Disk*.

You must be superuser to make the alterations explained below. Read through to the end of all steps before beginning the procedure.

Choose A Unique Client Name

Choose a name for the new client machine that is unique to the local network of machines. On the master yellow pages server, look in the file /etc/hosts to see what machine names are already used in your domain and choose a name that does not appear there.

If you've forgotten your master yp server you can find it out by typing:

```
% rsh `ypwhich` ypwhich
```

Edit Two Files On The Master yp Server

After you have chosen the name, stay on the master yp server and add a line for the new client to the master server's /etc/hosts file and to the /etc/ethers file. These entries enable the nd server to recognize the new client.

There are two parts to an /etc/hosts entry — the internet address and the host name. See the example below.

```
192.9.200.1 nancy
192.9.200.2 sluggo
192.9.200.3 taco
192.9.200.4 dutch
```

The last component of the internet address, the host number, must be unique for each client on the local network; assign an unused host number. Following the internet address, add the host name for the new client machine. For example, the new client machine here is 'dutch', whose internet address is 192.9.200.4; the host number is 4.

The /etc/ethers file format is similar to /etc/hosts. However, the ethernet address of the client is used instead of the internet address. For example, an /etc/ethers file looks like:

```
8:0:20:0:20:1    nancy
8:0:20:10:1b:2c  sluggo
8:0:20:10:0:3a   taco
8:0:20:0:2:6b    dutch
```

To find the client's ethernet address, power up the new client machine. To power up, switch on the Sun Workstation. As soon as the self-test completes, abort by typing `SETUP-A` or `L1-A` or `BREAK`, depending on your keyboard. Then read the Ethernet address from the PROM monitor's sign on messages. (*Installing UNIX On The Sun Workstation* describes the above process, in case you have forgotten the details.) The Ethernet address is a 6 byte hexadecimal value, each byte separated by a colon, for example:

```
8:0:20:1:e:29
```

Keep this number handy for the client machine, you will use it again in a minute.

Next, update the `yp` databases using `/etc/yp/make`:

```
# cd /etc/yp
# make hosts ethers
```

This pushes the new version of the database to the rest of the machines on the network, and makes sure the new client's `nd` server recognizes it.

Edit The `nd` Server's `/etc/nd.local` File

Now return to the `nd` server machine, where you must edit `/etc/nd.local`. A full discussion of this file, the command lines in it and their options, may be found above in the section *The Network Disk*. A terse description of the procedure to add a new client follows here.

You must put the client's Ethernet number in thend server's `/etc/nd.local` file. It's the same number you added to the `/etc/ethers` file above. If you lost the number, look in the section *Edit Two Files On The Master* `yp` *Server* for an explanation of how to get it.

Decide which method of assigning a partition you plan to follow. There are three ways to assign a new client to a soft partition on an `nd` server. In each you alter (or create in case 3) lines in `/etc/nd.local`. Remember, you will only use one of the ways described below.

1) You may assign the new client a never used soft partition. In this case, a new client receives an existing soft partition created by `setup` at installation. All partitions created by `setup`, even those not yet used, are shown in `/etc/nd.local`. Find the two appropriate `user` lines for the partition and edit in the new client machine name. Also make sure to update an `ether` entry for the new client machine. The section below, *Preparing An Unused Partition For A New Client*, more fully discusses this first case.

2) You may assign a previously used soft partition to the new client. That is to say, you may put the new client on the partition of a client removed from the server. In this case, find entries in `/etc/nd.local` that belonged to the former client. Change the former client machine name to the new client machine name on the `user` lines for the root and swap devices. On the `ether` line, change the name and give the new client's Ethernet address. The section below, *Preparing A Previously Used Client Partition*, more fully discusses this second case.

3) You may define a new partition in any unpartitioned space left on your network disk, and then assign it to a new client. In this case you first determine the new partition's size, then add two new `user` lines to the `/etc/nd.local` file. You also need to add an `ether` line for the new client. In many systems the root and swap partition sizes are standard for every client. In that case, use the standard *nblks* value for the new client's root and swap in your `/etc/nd.local` file — assuming of course enough room remains for a standard root and swap on the disk. The *startblks* value for any line is the sum of the *startblks* and *nblks* entries for the line preceding it. If you do not use a standard *nblks* value for partition size, you must make sure that the partition size you choose will place the partition boundary on a cylinder boundary. Follow these steps to make the determination:

   - Run `/etc/dkinfo`(8) to find out the number of sectors per track, and the number of tracks per cylinder on your disk.
   - Multiply sectors per track by tracks per cylinder to obtain the number of sectors per cylinder on your disk.
   - The number of sectors per cylinder is the minimum number of 512 byte blocks that can be allocated in the *nblks* field of the `user` line in `/etc/nd.local`. All *nblks* values greater than this minimum must be multiples of the number of sectors per cylinder.

   In order to correspond the *nblks* value to disk storage capacity, you can multiply *nblks* by 512 for an exact byte count. For an estimate in Mbytes, divide *nblks* by 2048.

   After you define a new partition on unused disk space (putting `user` lines for root and swap in `/etc/nd.local`) follow the procedures described below in the section *Preparing A Previously Used Client Partition.*

Any time you change the `/etc/nd.local` file, you must use it as input to the `/etc/nd` utility (which controls the network disk service of the kernel) to make it take effect:

```
# /etc/nd < /etc/nd.local
```

When you run this command after adding a new client, you will sometimes see an error message beginning:

```
nd user:  unknown host newclientname
```

This usually indicates that the machine serving the yellow pages to the nd server does not yet have an up-to-date version of the databases for the /etc/hosts file. See the section above on *Debugging A Yellow Pages Server* for instructions on how to update the databases.

Preparing An Unused Partition For A New Client

If, during installation of your system, you made extra client partitions with setup and are now ready to use one for the first time, follow these instructions. If you are putting a new client on a previously used partition, the procedure is different; go to the next section.

During the installation process you had the option in setup to add 'extra' client partitions at the end of the disk. If you exercised that option, some partitions were made and held in waiting for the system to use. These partitions are ready, with all the necessary files and links installed. On the nd server, follow the procedures below.

First mount the new client partition so you can work on it. Make sure a directory exists to mount it on; if not, make one. The name of a client partition is /dev/ndl*, where asterisk represents the value in the ndl# field of the user root line (the last field on the line) in the /etc/nd.local file. For example, user cairo, shown here in a sample user root line:

```
user cairo 0 /dev/xy0g 356040 10120 11
```

would be mounted by the following commands:

```
# mkdir /mnt     [if necessary]
# mount /dev/ndl11 /mnt
# cd /mnt
```

Once mounted, the /dev/ndl* client partition takes its name from the directory it is mounted on. So, in this example, all files in the cairo client partition are known as /mnt/*somefile*/... while mounted.

Now set up a number of administrative files on the new client's partition. They are:

1) /mnt/etc/passwd. setup provides a 'vanilla' password file that the system consults before using the password file from the yellow pages service. Add a line for the new client's primary user so s/he can login even when the yellow pages fails; see *Adding a New User* above. (CAUTION: Normally, you always use /etc/vipw to edit the password file. In this special case, use the regular editor to edit /mnt/etc/passwd. Using /etc/vipw now would bring the server's password file, rather than the new client's, into the editor.) One more note, the password file provided by setup does not have a root password. Tell the new client that root should get a password right away to prevent unwanted login on the new client machine.

2) /mnt/etc/group. Entries for this file will be provided by the yellow pages server if the client's file terminates in a line with the single '+'

character on it; otherwise you should move a copy of the nd server's /etc/group file to the client's partition.

3) /mnt/etc/printcap. Copy the nd server's version of this file to the client's partition:

```
# cp /etc/printcap /mnt/etc/printcap
```

4) /mnt/etc/hosts. When the client is up and running normally, the yellow pages provides this file. However, to enable the client system to come up single-user, there must be a version of the file containing an entry for *client_name* and localhost on the client's partition. For example, the minimum workable file for a client named 'pegasus' would be:

```
192.9.1.124 pegasus
127.0.0.1   localhost
```

Of course your internet numbers will be different. For details, see the section on editing the /etc/hosts file above.

5) /mnt/etc/networks. This can be provided by the yellow pages if the file contains a '+' line.

To enable the mail facility on the new client, you must copy one more file to the new client partition:

```
# cp /usr/lib/sendmail.subsidiary.cf\
/mnt/private/usr/lib/sendmail.cf
```

Note that you may enter the above command on a single line. This display wraps to a second line only because of page size limitations. If you do want to wrap your command, you must escape the carriage return with a backslash (\) as shown above.

In addition, edit the /mnt/etc/rc.boot file on the client partition, changing the line near the beginning which reads

```
/bin/hostname server_name
```

to

```
/bin/hostname new_client_name
```

Also check that the proper new client domain name is specified on the line

```
/bin/domainname domain_name
```

Now change out of the mounted directory and unmount the new client partition:

```
# cd
# umount /dev/ndl11
```

The procedure outlined above is one way to install a new client. There are certainly other ways to proceed if you are familiar with partitions and network disks. Beginners should follow the methods shown here.

Preparing A Previously Used Client Partition

When you give a used partition to a new client, you must remember two things: first, remove all customized or special features, files, or links the old client may have installed; and second, give the new client a generic partition exactly like `setup` gives clients during UNIX installation. There are many ways to do this. The simplest way we know is to begin with a 'copy' of a brand new partition. And the best way to get such a 'copy' is to `dump` a raw client partition to tape when it is brand new, before any work has been done on it. Of course, if you ignored the advice given during installation to do just this, you will have to prepare the partition 'by hand.' See *Installing UNIX On The Sun Workstation.*

If you have your 'copy' of a partition ready, do the following.

1) Make the new client's root partition with `/etc/mkfs`. (This destroys the old client's data, so save anything that could be wanted later.) The command is:

```
# /etc/mkfs /dev/rndl* nblks sectors heads 8192 1024
```

The italicized values (including the asterisk) must be filled in for your system. Get the proper * and *nblks* values from the new client's entry in `/etc/nd.local`. You also need to know the `sectors` and `heads` information for your disk. If you have forgotten it, you can run `/etc/dkinfo` again to determine it. See the earlier section on *Network Disk* for a full discussion of those procedures.

2) Now, mount the newly created partition and change directory to it.

```
# /etc/mount /dev/ndl* existing_directory
# cd existing_directory
```

3) Restore the `dump` tape into the current directory with `restore r`

```
# restore r
# rm restoresymtable
```

4) Unmount the restored directory and check the file system consistency.

```
# /etc/umount/ existing_directory
# /etc/fsck -p /dev/rndl*
```

5) You have now created a partition like the one that `setup` creates during system installation. No trace of the previous client remains on the file system. Thus, you should now go to the beginning of the section above, *Preparing A New Client Partition*, and proceed as if this were a new and unused partition created by `setup`.

Booting The New Client Machine

If you have successfully completed all the steps above, you can set the new client loose on her machine. The new client will need to boot the machine; for a discussion turn to the manual *Installing UNIX On The Sun Workstation*. After booting, the client should set up the network and mail facilities; for a discussion of those procedures see the related sections in this *System Administration Manual*, and in *Installing UNIX On The Sun Workstation*.

3

Disks And File Systems

Disks And File Systems **69**

3.1. Disk Device Terminology 69
Controllers, Device Drivers, Units 70
SCSI Disks vs SMD Disks 70
SMD Disk Formatting, Mapping, Slipping 71
SCSI/ST-506 Disk Formatting, Mapping, Slipping 72
Labels And Partitions 73
3.2. How UNIX Is Organized On The Disk 75
Some Basic Terms 75
The Major UNIX Directories 76
Server-Client File Systems And Partitions 80
3.3. Backing Up File Systems With `dump` 81
Reel Tape vs. Cartridge Tape 81
Backing Up Diskless Clients' `ndl` Partitions 82
Sample Scripts For File System Dumps 83
File System Dumps Over Ethernet 86
Other Files Relating To `dump` 86
3.4. Restoring Files With `restore` 87
Restoring One Or More Particular Files 87
Restoring An Entire File System 88
Restoring A Damaged 'root' Or `/pub` File System 89
Restoring Files Over Ethernet 91
3.5. Maintaining File Systems 93

Disk Capacity — Checking The Disk Resource Usage 93
The Disk Quota System 94
Making Room On File Systems 97
Files Needing Periodic Attention 98

3

Disks And File Systems

This chapter begins with a brief introduction to some terms that will help you understand how the disk storage devices work and how UNIX uses them. Following that we give an overview of UNIX organization on your disk(s) — what the major file systems are and how they are known by the hardware and software components of your system. (In addition, a paper explaining the UNIX file system check program, `fsck`(8), appears at the end of this manual in the *Tutorials* section).

After the introduction and overview, we discuss some practical aspects of data storage on disks. We explain how to backup (an absolute must) and restore the files that comprise UNIX, as well as those files users create to store their own data and programs. We also explain how to free up space on disks that have become full or nearly full. There are suggestions about what files to remove and what files to keep an eye on so they won't become unacceptably large.

3.1. Disk Device Terminology

This section introduces some of the terminology of disk drives.

Platters

Most common disk devices consist of a number of *platters* mounted on a spindle spinning at a high speed. Disks store information on a thin magnetic coating. Usually, they store information serially in bits of encoded magnetic impulses — a method similar to the way magnetic tape encodes information.

Heads

To read and write the information on the surface of the disk, a number of *heads* are mounted on a common arm so they travel together. Usually, there are two heads for each platter (one for the top and one for the bottom surface). The arm moves radially in and out over the surface of the disk, positioning the heads. This radial head movement is called *seeking*. The *seek* positions are controlled by the hardware of the disk drive.

Tracks and Cylinders

The portion of a disk passing under a single stationary head during disk rotation is called a *track*. At any seek position, a track passes under each head. The set of tracks for all the heads at a single seek position is called a *cylinder*, and is in effect a stack of rings one bit 'wide', equidistant from the spindle. To change from one cylinder to another, the heads must move inward or outward, but to change from one track to another in the same cylinder, the system just switches electronically to a different head.

Sectors

Each track is further divided into segments called *sectors*. To read successive sectors in the same track, the heads remain stationary as the disk spins under them. The disk drive generates a timing signal, and this signal identifies the sector for the disk controller.

The cylinders, tracks and sectors each inhabit a distinct dimension that separates and locates them: cylinders are located by distance from the spindle, tracks are separated from each other by surface location on platters, and sectors are separated from each other by time.

The sector is the basic unit of storage on the disk. On Sun systems, each sector contains 512 bytes of data, and is surrounded by a header and trailer containing addressing and error correction information. All of this information is laid out as a string of bits, which is grouped into bytes by the controller board. Thus, depending on its function, a program can treat the disk as a stream of bytes, as a series of 512 byte sectors, or it can 'know about' the cylinder/track/sector arrangement. In addition, the UNIX operating system introduces a further level of organization on the disk, and most programs running under UNIX treat the disk device as a *file system.*

Controllers, Device Drivers, Units

There are several layers of hardware and software between any program which uses the disk, such as `fsck`(8), and the disk drive itself. The hardware board which communicates directly with the drive is the *disk controller*. It takes care of many details of error checking, data transfer, and arrangement of the data on the disk. The software instructions to the controller board come from the *device driver*. It arranges for buffers, handles interrupts from the controller, handles disk errors, keeps track of requests for the disk from different programs, and takes care of other disk functions.

A disk controller may have more than one drive attached to it. The individual disk drives are called *units*. Inside UNIX the drives are referred to with a prefix that designates the type of controller. Thus, the first drive attached to a Xylogics controller would be referred to as `xy0`, the second as `xy1`, and so on. However, if there is more than one controller, the disks controlled by the same type of controller are numbered sequentially. With two drives on the first Xylogics controller, `xy0` and `xy1`, the first drive on the second controller would be `xy2`. The unit to which a particular disk drive responds is controlled by switches on the drive itself, and care must be taken to set these switches appropriately. Details for the various boards you might use are given in the *Hardware Installation Manual* for your Sun Workstation.

SCSI Disks vs SMD Disks

Sun currently uses two different disk interface technologies, SMD and SCSI/ST-506. For example, the SMD interface (Storage Module Disk) is used on a Fujitsu disk drive with a Xylogics disk controller; the SCSI interface (Small Computer System Interface) is used on a 71 MB Fujitsu and on a Micropolis disk drive. SCSI uses an adapter to translate between the system bus and a SCSI bus. The SCSI bus is a simple interface standard for communicating with peripherals. Sun uses the SCSI interface to communicate with the 1/4" tape drives and the Micropolis disk drives on the Sun 120/170, and Sun 160 products. The 1/4" tape drives for the 100/150 products have a controller which plugs directly into the

multibus.

The CPU makes requests for data from SMD controllers in the form: cylinder_#, track_#, sector_#. Thus, with the SMD interface, the system has to 'know about' the geometry of the disk. This allows UNIX to arrange the data on the disk so that different parts of files are positioned for efficient sequential access. On the other hand, this increases the computational load on the CPU for each disk access.

The SCSI bus treats the devices on it as idealized block devices. Requests across the SCSI bus ask for a block number from a specified device. Each device on the SCSI bus has a controller. For example, the controller for the Micropolis disk is an Adaptec, which receives commands from the CPU through the SCSI adapter and bus, and communicates with the disk across an ST-506 interface.

Because of these differences the SMD drives and the SCSI/ST-506 drives are handled differently by the system. When communicating with an SMD drive, the system makes requests of the form: cylinder_# track_# sector_#, while for the ST-506 disks, the system requests: block number nnnnnn. Then the SCSI controller translates the block number into a particular cylinder, track and sector. This idealized representation of the disk device cuts down on the CPU overhead for file transfers, but also makes it impossible for UNIX to optimize the layout of files on the disk.

SMD Disk Formatting, Mapping, Slipping

The arrangement of sectors on a track depends upon the configuration of the controller board and the disk drive. Furthermore, SMD drives distinguish between *physical sector numbers* and *logical sector numbers*. In addition to the platters which store data, a specially marked platter identifies the location of the different seek positions, enabling the heads to move to the appropriate cylinder in response to a seek command. This platter also has a timing mark which measures a full rotation of the disk platters.

Physical sector 0 follows this timing mark. It takes more time to shift from one head to another than it takes a sector to pass beneath the heads. Therefore, to facilitate sequential accesses to consecutive tracks within a cylinder, logical sector 0 of successive tracks is staggered. If the physical and logical sector numbers were not staggered, the drive might miss the beginning of the 0 sector of a track when attempting to read two sequential tracks; so logical sector 0 of track 0 of a cylinder is located at physical sector 0, logical sector 0 of track 1 is located at physical sector 1, and so on. The time required to seek from one cylinder to the next depends on the distance of the seek, but is great enough that this pattern need not be continued from one cylinder to the next.

Although the circuitry in the disk drive knows about the physical sectors to some extent, the controller writes a header on the disk identifying the beginning of a logical sector, as well as the number of the sector, the track, and the cylinder which it is in. The header also enables double checking of the disk drive electronics.

Following the header are 512 bytes of data and a trailer. The trailer has a code number derived from the contents of the data section of the sector. This code number can be used for error correction; the controller calculates it and writes

into the trailer. When the controller reads data back from a sector, it calculates a fresh code number from the data it reads. Then it compares the two code numbers and if they differ issues an error signal. These *Error Correction Codes* are calculated in such a way that certain types of errors can be fixed, and the data recovered. If the data can be reconstructed, the error is termed a `soft ECC error`; if it cannot, the controller issues a `hard ECC error`.

Between successive sectors, there is a gap with no data.

The process of writing out these headers and trailers, called formatting marks, is called disk formatting; `diag`(8) is the disk formatting program. `diag`(8) also performs one or more *surface analysis passes*, during which it writes out different patterns to the data areas of the disk, then reads them back and compares them to the original.

No disk is entirely perfect; there are usually spots where the magnetic coating is thin, or has a speck of dust embedded in it, and data cannot be properly recorded there. The surface analysis passes identify the sectors which have these defects. SMD controllers use five different surface analysis patterns, each designed to find a different sort of defect. Therefore, multiples of five surface analysis passes are preferable.

When `diag` finds a defective sector on an SMD drive, it identifies the bad sector by writing a special code into its header, and data is written into a different sector instead. There are two methods for choosing the alternate sector, depending on the disk drive. One is *mapping* the other is *sector slipping*.

Mapping substitutes a sector at the end of the disk for the sector with the defect, and makes an entry in a 'bad block map' which keeps a list of bad sectors and their substitutes. This method is known as 'mapping out' a sector. On the other hand, sector slipping reformats the track containing the bad sector and shifts the numbers of the logical sectors by one for all the subsequent sectors. Sector slipping is currently used on all the SMD drives that Sun ships. For an SMD controller to support sector slipping, there must be more physical sectors than logical sectors in a track. The controller then finds the actual location of a logical sector, so that slipping sectors is transparent to the system. Each method has its advantages and disadvantages, and is also dependent on the drive itself.

See the chapter on *Diagnostics* in this manual — the `diag` section — for further information about SMD disk formatting.

SCSI/ST-506 Disk Formatting, Mapping, Slipping

On the SCSI/ST-506 disks all the details of formatting, sector slipping, block location, and header checking are handled transparently by the ST-506 controller board. To achieve this transparency, the controller board has to 'know about' disk defects prior to formatting. Defect information can be read from the disk if the defect list has already been written on it. If the list has not been written on the disk, you may enter it by hand from the printed defect list supplied with the system.

When the formatting process reaches a track which appears in the defect list, it marks the defective area as bad and moves all the following sectors one sector further on. With the SCSI/ST-506 disks, sectors can be slipped across track and

cylinder boundaries, so all of the spare sectors are grouped together at the end of the disk. SCSI/ST-506 disks are always formatted according to the defect list which `diag` currently has in memory. Thus, if the list has been edited, you must use the current copy, and if there is no current copy, you must read the list in from the disk before formatting. It also pays to check the current version of the defect list against the printed version supplied with the system.

Surface analysis passes can be done on a formatted SCSI/ST-506 just like on an SMD disk. However, with the ST-506 the bad blocks are added to the copy of the defect list which `diag` has in memory rather marked on the disk. After surface analysis, the disk should be formatted again. With ST-506 disks, the formatting process only takes about 5 minutes, and while 5 surface analysis passes take between 20 and 30 minutes, it is worth the time.

See the chapter on *Diagnostics* in this manual — the `diag` section — for further information about SCSI/ST-506 disk formatting.

Labels And Partitions

In addition to disk format organization, which is largely for the benefit of the controller, there is a further, larger scale organization for the benefit of the device driver. This organization is called disk partitioning. Partitioning divides the disk into *partitions*, each of which corresponds to one of the device entries: `/dev/`*xx*`#`*a*; where *xx* is a controller type, such as `xy` for Xylogics, # is the unit number of the disk attached to the controller, and *a* is an alpha value from 'a' to 'h'. These partitions are not marked as such, but are described by a *label* written on cylinder 0, track 0, sector 0 of the disk. The label contains an entry for each of the eight partitions `a - h`. Each partition entry has an offset, which is the cylinder number for the first cylinder of that partition and a size for the partition expressed in blocks. The label also contains a description of the disk drive geometry: it specifies the type of disk, the total number of cylinders, the number of heads, and the number of logical sectors per track.

`diag` supplies a default partition for the disks which Sun sells with its systems. But if you know what the particular requirements of your system will be, you may customize disk partitioning when you install your system for the first time. For example, the table below shows the default partition for a Fujitsu 130 Megabyte disk.

Table 3-1 *Sample of Fujitsu 130 MB Disk Partitions*

Partition	Starting Cylinder	Size In Sectors
a	0	15884
b	50	33440
c	0	262400
g	155	212800

The disk driver knows about partition 'a' the root partition, partition 'b' the swap partition, and partition 'g' the rest of the disk. Partition 'c' covers the entire disk and allows the disk driver to address the entire disk. For example, partition 'c' is

often used on the second, third, or forth disk of a multi-disk system. It permits you to define a single large file system filling the entire disk.

The partitions divide the disk into functional units; each partition serves a different purpose. The various partitions on a disk are unambiguously referenced through the `/dev` entries for a disk. For further information about disk partitions, see the section, *Sun Network Disk Service* in the chapter called *Sun Network Services*.

Several backup copies of the label at the beginning of the disk are kept at the end of the disk — in case the primary label is corrupted. The label can only be written or restored with the `diag` program.

3.2. How UNIX Is Organized On The Disk

Some Basic Terms

The Sun UNIX operating system consists of the *kernel* and many hundreds of other files containing data and programs. At boot time the *kernel* is loaded into memory; it resides there until the system is shut down. Other files are loaded into memory as needed.

The kernel manages all of the physical resources of the workstation. Kernel functions include:

- Implements the *file system*, a tree-structured hierarchy of directories and files residing on disk or network disk or the network file system, and permits processes to create, read, write, and remove these files.
- *Virtual memory*, which allows up to 16 megabytes of unsegmented address space for each process by automatically moving 'pages' of information from disk to main memory as needed.
- The *scheduler*, which keeps track of all active processes and decides which gets to run next.
- *Device drivers*, software routines which control physical devices such as graphics display, mouse, keyboard, disk, tape, RS-232 serial ports, and Ethernet.
- *Networking software* which implements network functions such as the TCP/IP protocols and the network disk (*nd*) function which supports diskless workstations.
- *Interprocess communication* facilities such as signals and sockets.
- Facilities for creating, examining, and modifying processes.
- System management functions, such as halting, booting, and error handling.
- Miscellaneous functions which make system resources (like memory, timers, etc.) available to processes.

The kernel resides on the disk in a single file, typically known as `/vmunix`. (As you will learn, there may be other kernels with other names). We mentioned the many hundreds of other files comprising the UNIX system, most of these are *commands*, also called *utilities*, programs the user invokes directly. Other files contain *libraries*. These are collections of software routines which may be selected from the library and incorporated into another program. Libraries form an extension of the basic system features implemented by the kernel. Many files contain *data* used by the programs in other files.

The *Commands Reference Manual for the Sun Workstation* describes the commands available. Sections 1, 6, and 7 describe user commands including games and demos; section 8 describes the system administration and maintenance commands. The *System Interface Manual* describes system calls, library routines, special files, and file formats. Section 2 describes kernel calls or 'system calls;' section 3 describes library routines; section 4 describes the special files used for devices; section 5 explains some of the most important data files. The

introductory section of each manual contains a useful summary of its contents.

The Major UNIX Directories

The UNIX file system is organized as a tree-structured hierarchy of directories, device nodes, symbolic links, and ordinary files. Although these objects could reside anywhere in the hierarchy, many UNIX commands expect certain files and devices to reside in specific directories, and will not function properly unless they do.

Following is a list of some of the directories used by UNIX and a brief description of the function of each one. You can examine their contents in more detail by browsing through the file system on a Sun Workstation, using the `cd` and `ls` commands.

`/` ('root')	The 'root' of the file system tree. It contains the kernel (`/vmunix`) and the important directories `/etc`, `/bin`, `/dev`, `/lib`, `/tmp`, `/stand`, `/lost+found`, `/mnt`, `/pub`, `/private`, and `/usr`, all described below.
`/etc`	The 'et cetera' directory contains various commands and data files, primarily those used for system administration. Many of these commands and files are described in the *System Interface Manual* or the *Commands Reference Manual*; in particular, sections 5 and 8 of those manuals. See also `/usr/etc` (below).
`/bin`	This is one of the three major UNIX directories containing user commands. The commands in `/bin` are critical; it is mounted when you are running single-user. These commands are typically invoked when a user types in the command name. The other two major user command directories are `/usr/bin` and `/usr/ucb`, described below. Conceptually, the commands in `/bin` are the most important and most commonly used. The name 'bin' originally derived from 'binary' because the files in `/bin` were binary (executable code) files, although currently several are ASCII shell scripts.
`/dev`	The 'device' directory contains all of the *device special files*, also known as *device nodes*. It also contains a shell script called `MAKEDEV`; when executed, `MAKEDEV` creates device nodes for all Sun-supported devices.
`/lib`	The 'library' directory contains files crucial to the C and Fortran compilers, including portions of the compilers themselves and the standard libraries for each language. (See also `/usr/lib`)
`/tmp`	Various UNIX utilities, such as `vi` and `ar`, create temporary data files in `/tmp`. Users may also use `/tmp` as a temporary workspace. Every time the system is rebooted, the script `/etc/rc` (see `rc`(8)) removes all files in `/tmp` except for subdirectories.

`/stand` See `/usr/stand` for a standalone machine and `/pub/stand` for an `nd` server or client machine.

`/sys` See `/usr/sys`.

`/lost+found` `fsck`(8) uses `/lost+found` as a repository for unreferenced files it finds on the disk. Normally this directory remains empty, but some files may appear in it after running `fsck` on a damaged file system. See also `/usr/lost+found` (below) and `fsck`(8).

`/mnt` Normally empty, `/mnt` is typically used to mount file systems temporarily.

`/private` On diskless clients and `nd` servers, `/private` contains certain files which would otherwise appear in the shared file system `/pub` (see below), but which must differ from one machine to another.

For example, the directory `/usr/lib` (see below) is shared by server and clients. However, the file `/usr/lib/aliases` is itself a symbolic link to `/private/usr/lib/aliases`, and a separate copy of it exists on each machine.

`/private/usr/adm`
This directory contains system accounting files. See `ac`(8) and `sa`(8) for more details.

`/private/usr/preserve`
When one of the editors (`vi`, `ex`, `edit`, or `e` — which is actually one editor running in four different modes) is interrupted without having a chance to exit normally (for example, in a system crash) the editor attempts to save as many of the user's changes as possible. These can be recovered later using the `vi -r` command. The `/private/usr/preserve` directory is used to preserve the necessary temporary edit files, pending recovery.

`/private/usr/spool`
This contains a number of subdirectories which may be characterized as `spool directories`. Spool directories are repositories for files pending further processing, typically by a program other than the one which placed the file into the spool directory. For example, `/private/usr/spool/lpd` holds files printed by the `lpr` command pending output to the printer; `/private/usr/spool/mail` contains users' 'system mailboxes,' which contain incoming mail pending reading by the recipient; and `/private/usr/spool/uucp` contains `uucp` data files and work files in transit to or from other machines.

`/private/usr/tmp`
: Like `/tmp` (see above), but `/private/usr/tmp` is available to users for their own purposes, rather than by system utilities like `vi`. Unlike `/tmp`, this directory is not cleaned up by /etc/rc upon reboot.

`/pub`
: On disk servers and their clients, a few files are moved from their usual place into a shared, read-only, 'public' file system mounted on the directory `/pub`. These are typically files required for clients to boot, including `/vmunix`.

`/pub/stand`
: On an `nd` server or a client machine, this directory contains diagnostic programs, many of which must be booted (standalone) rather than run as UNIX processes. For example, the `diag` utility, which formats and tests disks, may be booted from a disk which already contains a UNIX file system by typing `b /pub/stand/diag` to the PROM monitor. Some of the programs in `/pub/stand` are currently undocumented and intended only for the use of Sun field engineers.

`/usr2/`*`user_name`*
: Under NFS, each user's home directory is normally a subdirectory of `/usr2`, and has the same name as the user's login name. See the section in this manual *Adding A New User To A System.*

`/usr`
: Contains many important sub-directories described below.

`/usr/bin`
: One of the three major directories containing user commands (see also `/bin` and `/usr/ucb`). The commands in `/usr/bin` are usually not as important as those in `/bin`.

`/usr/crash`
: When rebooting after a UNIX system crash or 'panic', the kernel core dump copies files into this directory and to make them available for further analysis. Note that on `nd` servers and clients, the crash will be copied to `/usr/crash/`*hostname* of the host machine. See the chapter on *Periodic Maintenance* in this manual for further information.

`/usr/demo`
: This directory contains some Sun graphics demo programs. See section 6 of the *Commands Reference Manual* and the manual *Installing UNIX On The Sun Workstation* section on how to load demos from the boot tape.

`/usr/dict`
: Contains English language spelling lists used by `spell`(1).

`/usr/etc`
: Like `/etc` (see above), this directory contains additional files used for system administration and maintenance. The files in `/usr/etc` are usually not as important as those in `/etc`.

`/usr/games`
: Contains games. See section 6 of the *Commands Reference Manual* and the manual *Installing UNIX On The Sun Workstation* section on how to load games from the boot tape.

`/usr/hosts` Contains a script called MAKEHOSTS, which creates, for each host in `/etc/hosts`, a symbolic link to the `rsh` command which is the hostname itself. For example, after running MAKEHOSTS and adding `/usr/hosts` to your '$path' shell variable, you can just type `sundial` instead of typing `rlogin sundial`.

`/usr/include`
This directory contains all of the standard 'include files' or 'header files' used to compile C programs. These files, whose names conventionally end with `.h`, contain definitions of useful constants and macros. Particular `.h` files are often explained by entries in sections 2 and 3 of the *System Interface Manual.*

`/usr/lib` This directory contains over a hundred files used by UNIX utilities. More varied than any of the directories described here, it is kind of a 'catch-all' for UNIX utility files that didn't quite belong anywhere else.

These include libraries (names ending in `.a`, see `/lib` above for description) supporting Pascal, SunCore, Sunwindows, and other system functions; programs used by various system utilities (for example, `f77`, `lint`, `lex`, `spell`, etc.); `troff` macros (`tmac`, `me`, `ms`); Pascal and Fortran; line printer filters; and more.

`/usr/local` This directory, not on all machines, is not a standard part of UNIX, but the system administrator often creates it to hold commands, programs, or other files. These might be obtained from third parties (such as the Sun User Group) and added to the system after installation. Usually, every user adds `/usr/local` (and/or `/usr/local/bin`) to his or her '$path' shell variable to take advantage of these local commands.

`/usr/lost+found`
If `/usr` is a separate, mounted file system, this directory serves the same purpose as `/lost+found` (see above) does for the root file system.

`/usr/man` Holds the on-line manual pages. See `man`(1), as well as the manual *Installing UNIX On The Sun Workstation*, the section on installing manuals, demos, and games.

`/usr/mdec` Contains boot programs and a script for installing them. Usually not touched after system installation, except sometimes when restoring a damaged root or `/pub` file system.

`/usr/pub` The pub here stands for publishing and has nothing to do with the `/pub` directory above. Do not confuse the two. `/usr/pub` contains miscellaneous tables including a table of the ASCII characters and tables used by the `nroff` and `troff`

text formatters.

`/usr/sccs` Contains commands used by `sccs`, the Source Code Control System. See `sccs`(1) and the *SCCS* tutorial in *Programming Tools for the Sun Workstation.*

`/usr/src` Not on all machines. On machines licensed to contain source code, this is where it lives. If you load the sources for the window system from the optional software, they go in this directory.

`/usr/stand` On a standalone machine, this directory contains diagnostic programs, many of which must be booted (standalone) rather than run as UNIX processes. For example, the `diag` utility, which formats and tests disks, may be booted from a disk which already contains a UNIX file system by typing `b /usr/stand/diag` to the PROM monitor. Some of the programs in `/usr/stand` are currently undocumented and intended only for the use of Sun field engineers.

`/usr/sys` The 'system' directory contains all of the files necessary to build or reconfigure the kernel. Kernel configuration is explained in the manual *Installing UNIX On The Sun Workstation.* When you set up a server and diskless clients, the `/usr/sys` directory is not copied to the clients' file systems. This saves over a megabyte of disk space per client. When necessary, you can construct kernels for both server and clients on the server.

`/usr/ucb` The third major directory containing user commands (see also `/bin` and `/usr/bin`). Most of the commands here originated at the University of California at Berkeley, hence the initials 'ucb'.

Server-Client File Systems And Partitions

The majority of the directories described above are shared by an `nd` server and its clients. The server and each client have separate `/etc` and `/bin` directories so they can run in single-user mode, as well as a separate `/private` for specific files.

The server-client environment is explained in more detail in the section on the *The Network Disk* in the chapter called *Sun Network Services.* Typically, the following things are true under NFS:

- A standalone machine, by default, has three partitions — `a` is 'root,' `b` is 'swap,' and `g` is `/usr`.
- A server machine has the following standard partitions — `a` is 'root,' `b` is 'swap,' `d` is `/usr`, `e` (if requested in `setup`) is `/usr2`, and `g` is `/pub` plus the client partitions.
- On a client machine, there are two `nd` partitions — `nd0` (the entire root file system) and `nd1` (the swap partition).

3.3. Backing Up File Systems With dump

The need for frequent and regular file system backups is discussed in the chapter *Periodic Maintenance* in this manual. Here we explain the mechanics of backups with dump(8). Before reading further, look at the dump page in the *Commands Reference Manual* to familiarize yourself with options mentioned below but not explained in great detail.

Reel Tape vs. Cartridge Tape

Disk files are backed up onto tape at regular intervals to preserve information that could be lost if disk problems occur. Your workstation will use either 1/2 inch reel or 1/4 inch cartridge tape, depending on the type of tape controller you have installed. A tape controller that writes high density tapes is available for 1/2 inch reel tape — but only with a Xylogics 472 controller. If you have this optional high density configuration, you can write more bits per inch (6250 vs. 1600) to tape and store more information on each tape. In addition, there are two available 1/4 inch tape controllers — the Sun archive tape controller and the SCSI tape controller. Both 1/4 inch tape controllers write at the same density. In general, the dump command works the same with every type. But, as you might expect, you must specify different device names and, in some cases, different flags for the different types of tape controller.

A generic command to dump to tape looks like:

```
# /etc/dump flags tape_device_name filesystem_to_dump
```

The *tape_device_name*'s for the controllers mentioned above are: mt0, 1/2 inch reel 1600 bpi; mt8, 1/2 inch reel 6250 bpi; ar0, 1/4 inch archive; and st0, 1/4 inch SCSI. So, specifically, the command to dump a 1/2 inch reel tape at 1600 bpi (mt0) looks like:

```
# /etc/dump flags /dev/rmt0 filesystem_to_dump
```

And the command to dump a 1/2 inch reel tape at 6250 (mt8) looks like:

```
# /etc/dump flags /dev/rmt8 filesystem_to_dump
```

While the command to dump a 1/4 inch archive tape (ar) looks like:

```
# /etc/dump flags /dev/rar0 filesystem_to_dump
```

And the command for a 1/4 inch SCSI tape (st) looks like:

```
# /etc/dump flags /dev/rst0 filesystem_to_dump
```

In each of the dump lines for 1/4 inch cartridge tape (and only for cartridge tape!) you must include the 'c' flag — which tells dump it is using a cartridge tape. Additionally, in the case of cartridge tapes, you should not use the default blocking factor; the dump would take far too long. Use the 'b' flag and give a blocking value of 126. (Note that you must restore a tape with the same blocking factor at which you dumped it.) For example, to do a level 9 dump on a SCSI tape of the file system xy0g blocked at 126, and upon completion write a message to the record file /etc/dumpdates, you would type the following:

```
# /etc/dump 9ucbf 126 /dev/rst0 /dev/rxy0g
```

Note that *flags* are grouped together without intervening spaces. We recommend always dumping the raw device (here rxy0g), it goes much faster.

If you do not want the tape to rewind after finishing the dump specified in a single `dump` command, add an 'n' to the *tape_device_name* argument, for example: `/dev/nrst0`. In this way you can get more than one file system dumped onto a single tape.

The generic command to rewind a tape and take it off line is:

```
# mt offline
```

You may type that to rewind all 1/2 inch reel tapes. With a 1/4 inch tape you need to add a flag to tell the system to use one of the 1/4 inch tape devices instead of the default 1/2 inch device. The commands are, for Sun archive:

```
# mt -f /dev/rar0 offline
```

and for SCSI:

```
# mt -f /dev/rst0 offline
```

Tape capacity varies with tape size and density. A 2400 foot, 1/2 inch tape at density 1600 bpi will hold approximately 30 MBytes of disk storage; while the same 2400 foot 1/2 inch tape at 6250 bpi will hold approximately 110 MBytes. A 1/4 inch tape (either archive or SCSI) holds approximately 20 MBytes. You need to know tape capacity when using `dump`'s no rewind option (`/dev/n`*device*) to dump several file systems onto a single tape. With no rewind, `dump` will not carry over onto a subsequent tape when the tape reel ends in the middle of a file system. (**NOTE:** This is not a problem when you begin a dump at the beginning of a tape reel. `dump` can and will carry over onto subsequent tapes, but only when the dump starts at the beginning of a reel. Thus there is no problem with a large dump if it begins a fresh tape.) Therefore, before dumping multiple file systems onto one tape of any type, you must calculate the sum, in megabytes, of all the file systems to be dumped, and parcel them out to make sure they will not overrun the end of the tape. You can obtain the size, in kilobytes, of the file system(s) you plan to dump with `df`(1).

Backing Up Diskless Clients' `ndl` Partitions

On a server the `/pub` file system and diskless clients' file systems are allocated as 'soft' partitions within larger 'hard' partitions. Remember that `/etc/nd.local` does this — see *The Network Disk* section in this chapter. On a server with one disk, the public and client partitions are usually 'soft' partitions of partition 'g' of disk '0'; for example, `/dev/xy0g`.

However, if you `dump` device `/dev/xy0g` on the server, you will only dump the `/pub` file system, not any of the `ndl` client file systems. To dump the client file systems, or run `fsck` on them, or whatever, you must treat them as distinct disk partitions. They are known on the server as `/dev/ndl0`, `/dev/ndl1`, and so on. The last digit on the `user` line in `/etc/nd.local` (if non-negative) is the `ndl` device number of the corresponding client partition. The lines ending in `-1` are usually paging areas, not file systems you want to dump. If your system included the lines shown below in the sample `/etc/nd.local` file, you would have to dump `/dev/rndl0`, `/dev/rndl1`, and `/dev/rndl2`, to backup clients 'bill', 'debby', and 'joan'. (**Note:** the designations `/dev/ndl0` and `/dev/rndl0`, or in other cases, `/dev/xy0g` and `/dev/rxy0g`, refer to the same device; `ndl0` is the way to refer to the blocked

device, and `rndl0` refers to the raw device. Raw devices are always dumped, see the examples below.)

```
user bill 0 /dev/xy0g 108560 39560 0
user bill 1 /dev/xy0g 148120 12880 -1
user debby 0 /dev/xy0g 161000 39560 1
user debby 1 /dev/xy0g 200560 12880 -1
user joan 0 /dev/xy0g 213440 39560 2
user joan 1 /dev/xy0g 253000 12880 -1
```

If possible, halt the client machine during incremental dumps (at least avoid dumping any heavily used machine). Always make sure the client is halted during level zero dumps and when `fsck` runs on its partition.

For more information about client partitions, see the section *The Network Disk* in the chapter *Sun Network Services*.

Sample Scripts For File System Dumps

The example shell scripts given below can be used for doing routine backup dumps of the file systems on disk servers and, where applicable, on standalone systems.

Before doing a level zero dump on any file system be sure to run `fsck`(8), the UNIX file system check program. This insures that a file system has no discoverable inconsistencies before it is dumped to tape. `fsck` is a multi-pass file system check program. Each file system pass invokes a different phase of the program. In successive passes, `fsck` checks blocks and sizes, path-names, connectivity, reference counts, and the map of free blocks (possibly rebuilding it), and performs some cleanup. For further information see `fsck`(8) in the *Commands Reference Manual*, and the *FSCK Tutorial* in the tutorial section of this manual. You should also run `fsck` whenever you have restored a complete file system (see the section below on `restore`). Remember to address the `ndl` partition properly when using `fsck`.

In the examples below we give one example for 1/2 inch tape and another for 1/4 inch tape.

In these examples, the `n` preceding the drive type indicates no rewind after the file system has been dumped. The `mt offline` rewinds the tape, and takes it off line. The `r` in the `rxy0a` (and in the other `rxy` and `rndl` file system names) indicates that the raw device should be dumped. We recommend that you always dump the raw device since it dumps considerably faster than the block device (`xy` or `ndl`).

The first example works on a server or standalone machine. It performs a full dump of a root file system, and level 9 incremental dumps of three other file systems: partitions '`e`', '`f`', and '`g`'. On a server, partition '`g`' would typically contain `/pub`; on a standalone machine, partition '`g`' would typically contain `/usr`. On a server machine, under NFS, partition '`e`' typically contains `usr2`. In each case, partition '`f`' is locally defined. These dumps are written consecutively on a single volume of tape. This script does not take arguments; it has been determined that the dumps of these file systems will fit on the single volume of tape.

For 1/2 inch tape drives (substitute for *drive* one of the following — `rmt0` for density 1600 bpi; `rmt8` for density 6250 bpi):

```
#
# Example 1: fixed incremental & root zero for 1/2 inch tape
#
/etc/dump 0uf /dev/ndrive /dev/rxy0a
/etc/dump 9uf /dev/ndrive /dev/rxy0e
/etc/dump 9uf /dev/ndrive /dev/rxy0f
/etc/dump 9uf /dev/ndrive /dev/rxy0g
echo DONE unloading tape
mt offline
```

For 1/4 inch tape drives (substitute for *drive* one of the following — `rar0` for archive tape controller; `rst0` for SCSI tape controller.):

```
#
# Example 1: fixed incremental & root zero for 1/4 inch tape
#
/etc/dump 0ucbf 126 /dev/ndrive /dev/rxy0a
/etc/dump 9ucbf 126 /dev/ndrive /dev/rxy0e
/etc/dump 9ucbf 126 /dev/ndrive /dev/rxy0f
/etc/dump 9ucbf 126 /dev/ndrive /dev/rxy0g
echo DONE unloading tape
mt -f /dev/drive offline
```

Assume this script is an executable file called 'example.1'. After mounting the tape you type:

```
# example.1
```

No arguments are allowed; it simply dumps the designated file systems each time it runs. You would not typically want to make a script like this to do level 0 dumps of server file systems other than root. Those file systems tend to be quite large and the full dumps are usually done 'by hand.' Of course, you could use a script in the full dump case, but it would dump just one file system, starting on a fresh reel of tape.

The next example does a level 0 dump of the specified list of `ndl` partitions passed to the script as arguments at execution time. This example provides flexibility to choose different diskless client partitions each time you run it. The `n` preceding the drive type indicates that the tape will not rewind after each dump and the files will be written consecutively onto the end of the tape volume, just as in Example 1.

For 1/2 inch tape drives (remember to substitute for *drive* just as above):

```
#
# Example 2: level zero for chosen clients on 1/2 inch tape
#
for i in $*
do
date
echo dumping ndl$i
/etc/dump 0uf /dev/nd<drive> /dev/rndl$i
done
echo DONE unloading tape
mt offline
```

For 1/4 inch tape drives (remember to substitute for *drive* just as above):

```
#
# Example 2: level zero for chosen clients on 1/4 inch tape
#
for i in $*
do
date
echo dumping ndl$i
/etc/dump 0ucbf 126 /dev/nd<drive> /dev/rndl$i
done
echo DONE unloading tape
mt -f /dev/<drive> offline
```

Assume this script is an executable file called 'example.2'. After mounting the tape type:

```
# example.2  n n n ...
```

The *n* arguments passed at execution are taken from the `/etc/nd.local` file on the server. In that file look at the end of the `user` lines for the clients you want to dump. For example, if `/etc/nd.local` looks like:

```
user joan 0 /dev/xy0g 213440 39560 2
user joan 1 /dev/xy0g 253000 12880 -1
user avb 0 /dev/xy0g 265880 22080 3
user avb 1 /dev/xy0g 287960 39560 -1
```

Then,

```
# example.2 2 3
```

would dump the partitions joan and avb. You must first determine that all the partitions dumped will fit on the tape volume — use `df` as mentioned above in the section *Reel Tape vs. Cartridge Tape.*

The final example performs level 9 incremental dumps for predetermined `ndl` partitions, 0 through 4 in the example script. The script takes no arguments, but you must predetermine that all the partitions will fit on one volume of tape.

For 1/2 inch tape drives (remember to substitute for *drive* just as above):

```
#
# Example 3: fixed incremental partitions for 1/2 inch tape
#
for i in 0 1 2 3 4
do
date
echo dumping ndl$i
/etc/dump 9uf /dev/n drive /dev/rndl$i
echo FINISHED unloading tape
mt offline
```

For 1/4 inch tape drives (remember to substitute for *drive* just as above):

```
#
# Example 3: fixed incremental partitions for 1/4 inch tape
#
for i in 0 1 2 3 4
do
date
echo dumping ndl$i
/etc/dump 9ucbf 126 /dev/n drive /dev/rndl$i
echo FINISHED unloading tape
mt -f /dev/n drive offline
```

File System Dumps Over Ethernet

You may dump files or partitions from a machine where you are logged in as superuser, onto a tape mounted on a remote machine's tape drive. The name of the machine you are dumping must be in the file `/.rhosts` on the remote machine. While superuser on the machine from which you want to dump files, type:

 # /etc/dump 9uf *remote_machine_name*:**/dev/***drive* **/dev/***device*

For 1/4 inch tape drives, remember to use the `b` and `c` flags as described above.

Remember to substitute the appropriate values for *remote_machine_name*, *drive*, and *device*.

Of course, you may request a different level dump from the one shown here. See `dump`(8) for more about options.

Other Files Relating To `dump`

If you specify the `u` option of `dump` (as in the examples above), a record of the date and level of the last successful full and incremental dumps for each file system or partition will be written to the file `/etc/dumpdates`. It is a readable ASCII file; you can look at it to verify the history of dumps on your system.

The file `/etc/fstab` (`fstab`(5)) contains static information about file systems. Commands read it and it should be kept up to date by the system administrator. It describes the file systems and swapping partitions on the local machine.

3.4. Restoring Files With `restore`

`restore`(8) restores files from tapes created by `dump`(8). You may use `restore` to reload an entire file system hierarchy from a level zero tape and incrementals that follow it, or to restore one or more single files from any `dump` tape.

Always make sure you restore from the correct tape. After you mount a tape, type `restore t` to read the tape creation date and other information.

For a complete discussion of `restore`'s functions and the options to those functions, see the `restore`(8) page in the *Commands Reference Manual*. Below we discuss three typical uses of `restore`.

Restoring One Or More Particular Files

Commonly, a user will lose a file or files through carelessness, a faulty program, or the improper use of a program. Cases like this require that the system administrator be able to restore individual files or small groups of files.

First find the tape on which the missing file(s) is likely to have been dumped in a state that most suits the user's needs. Note that this is not necessarily the most recent backed up version of the file. Sometimes a user wants last week's program that she forgot to save a copy of before she began to make this week's 'improvements.' Keep familiar with the storage and organization of backup tapes at your site so you can locate tapes months or years old. It becomes particularly important when you must restore a subdirectory containing many files worked on over a long period of time. Frequently, in a case like this, the needed versions of files are not on a single dump tape.

You should almost never restore a file to its original directory. Either make a new directory to hold restored file(s) (and make sure that the user owns it), or change directory to `/usr/tmp` and restore the file(s) there. The name(s) of the restored file(s) is the name on the tape (the full pathname from the root of the dumped file system) relative to the current working directory at the time of the restore. The system administrator should probably let the user change the restored file to whatever name he or she chooses, since a currently existing file in the user's directory may bear the same name as the old restored file. The system administrator can send mail to the user, something like:

```
A copy of your file "john/prog.c"
from a dump tape of June 8, 1984
was restored as "/usr/tmp/john/prog.c"
```

After files have been restored, check to make sure ownerships and permissions have stayed as they are on the dump tape (or have been changed if you want to change them).

Below are three `restore` options particularly useful when restoring small numbers of files, or an entire directory and the subdirectories under it. See the `restore` *Commands Reference Manual* entry for all function options.

- `restore t` *filename(s)* — lists the names of specified files if they occur on the tape; useful for quickly finding out if a certain file(s) is on a particular tape. Without the *filename* argument, it lists the entire hierarchy of the tape.
- `restore x` *filename(s)* — restores the named file(s) from the tape. If *filename* is a directory, it recursively extracts the directory.

- `restore i` — You can restore files interactively with the `i` function. It sets up a shell-like environment that provides great flexibility for selecting files to restore. Using `restore i`, you may look through an entire directory while you search for particular files to add to the restore list.

Restoring An Entire File System

Sometimes a file system becomes so completely twisted that you have to restore it entirely. This can be time consuming and annoying, but you won't lose too much data if you have a complete set of backup tapes — even if a faulty head has just eaten a whole disk. If you have done backups properly, you should be able to restore the system to its state as of the previous working day. (In any event, certainly up to the last incremental dump, one of which should be done before or after every working day.)

Let's assume a major disaster occurred on a file system, and you have corrected the software or hardware problem to the point where you're ready to restore from the dump tapes. **NOTE:** if you are restoring an entire 'root' (`/`) or `/pub` file system, skip to the section below; it includes special actions which must be taken for those particular systems. Continue here for other cases. Of course, you must be superuser to perform these operations. Do the following:

1) Recreate the damaged file system. For a hard partition, do the following:

   **`# /etc/newfs /dev/r`*device*

   where *device* is the `/dev` for the hard partition you will restore, for example `xy2h`. Note that you address the raw device with `r`. (You may use `/etc/mkfs` with hard partitions. However it means more work; you must supply information that `/etc/newfs` finds for itself. If you do use `/etc/mkfs` for some reason, you can obtain the necessary information from `dkinfo`(8).)

   For a soft (or client) partition, do the following:

   `# /etc/mkfs /dev/r`*device nblks sectors heads* `8192 1024`

   where:

   - *device* is the `/dev` entry for the soft partition you are restoring, for example `nd12`. Note that you address the raw device with `r`.
   - *nblks* is the size of the partition in 512 byte blocks. For a client partition, obtain this value from the `size` field in `/etc/nd.local`.
   - *sectors* is the number of sectors per track on the physical disk.
   - *heads* is the number of heads per sector on the physical disk. The information for *sectors* and *heads* varies from one configuration to another, and can be obtained from `/etc/dkinfo`(8).

2) Check the file system:

   **`# /etc/fsck /dev/r`*device*

   Remember to use the raw device.

3) Mount the file system on an existing directory prior to reloading. In this example `/mnt` already existed. Use the block device, not the raw device here. **Note:** If this is a client `ndl` partition, make sure the client machine is halted.

```
# /etc/mount /dev/device /mnt
```

4) Change to the `/mnt` directory and restore the level zero tape.

For the 1/2 inch tape drives (substitute for *drive* one of the following — `rmt0` for density 1600 bpi; `rmt8` for density 6250 bpi):

```
# cd /mnt
# restore rvf /dev/rdrive
```

And for 1/4 inch tape drives (substitute for *drive* one of the following — `rar0` for archive tape controller; `rst0` for SCSI tape controller):

```
# cd /mnt
# restore rvbf  blksize /dev/rdrive
```

Continue to restore incremental tapes in the order of lowest level number first, working up to the most recent incremental tape last.

5) Remove a file that `restore` creates in the current working directory during its operation; it is called `restoresymtable`.

```
# rm restoresymtable
```

6) Unmount the file system and check it again with `fsck`. Remember to check the raw device by typing **r***device*.

```
# cd
# /etc/umount /dev/device
# /etc/fsck /dev/rdevice
```

7) Do a full dump (level zero) of the newly restored file system.

For a 1/2 inch tape drive:

```
#  /etc/dump 0uf /dev/rmt0 /dev/rdevice
```

And for 1/4 inch tape drives:

```
# /etc/dump 0ucbf 126 /dev/rdrive /dev/rdevice
```

Substitute for **r***drive* one of the following — `rar0` for archive tape controller; `rst0` for SCSI tape controller.

Restoring A Damaged 'root' Or `/pub` File System

Restoring a damaged 'root' or `/pub` file system from a dump tape presents a special case because the utilities needed are in the damaged file system. The general strategy is to load and boot the mini-UNIX file system from the boot tape, and reload the file systems from there.

Specifically, for a 'root' file system do the following:

1) Locate and fix any bad blocks or other hardware problems.

2) Load and boot the mini-UNIX file system from the boot tape according to the instructions in the manual *Installing UNIX On The Sun Workstation.* If disk problems have occurred, boot `diag` from tape and fix the disk. Be sure to check for a good disk label using `diag`'s `verify` option.

3) Make the entries in `/dev` on the mini-UNIX file system for your tape and disk. The devices are `mt` for a 1/2 inch tape, `st` for 1/4 inch SCSI tape, and `ar` for 1/4 inch Archive tape; for disks they are `xy` for Xylogics disk, `sd` for SCSI disk, and `ip` for Interphase disk. Look in `MAKEDEV` in `/dev` to find the device names for your system's tape and disk types, and read `MAKEDEV`(8) in the *Commands Reference Manual.*

```
# /dev/MAKEDEV device
```

4) Recreate the 'root' file system:

```
# /etc/newfs /dev/rdisk_device0a
```

Use `r` for raw device, and be sure to copy down and save the backup super block numbers that are returned.

5) Check the root file system:

```
# /etc/fsck /dev/rdisk_device0a
```

6) Mount the file system so it can be reloaded. The example assumes the prior existence of the `/mnt` directory; make it with `mkdir` if it does not exist.

```
# /etc/mount /dev/disk_device0a /mnt
```

7) Change to the `/mnt` directory and restore the level zero tape.

For the 1/2 inch tape drives (substitute for *drive* one of the following — `rmt0` for density 1600 bpi; `rmt8` for density 6250 bpi):

```
# cd /mnt
# restore rvf /dev/rdrive
```

And for 1/4 inch tape drives (substitute for *drive* one of the following — `rar0` for archive tape controller; `rst0` for SCSI tape controller):

```
# cd /mnt
# restore rvbf  blksize /dev/rdrive
```

Continue to restore incremental tapes in the order of lowest level number first and working up to the most recent incremental tape last.

8) Remove a file that `restore` creates in the current working directory during its operation; it is called `restoresymtable`.

```
# rm restoresymtable
```

9) Unmount the file system and check it again with `fsck`. Remember to check the raw device using `r`.

```
# cd
# umount /dev/disk_device0a
# /etc/fsck /dev/rdisk_device0a
```

10) Do a full dump (level zero) of the newly restored root file system.

For a 1/2 inch tape drive:

```
# /etc/dump 0uf /dev/rmt0 /dev/rdevice0a
```

And for 1/4 inch tape drives:

```
# /etc/dump 0ucbf 126 /dev/rdrive /dev/rdevice0a
```

Substitute for *rdrive* one of the following — `rar0` for archive tape controller; `rst0` for SCSI tape controller.

If this is a standalone system, or a server on which the `/pub` partition does not also need to be restored, you can reboot the system at this point.

If you are on a standalone system, and `/usr` has been damaged, you can restore it after booting single-user by following steps 4 through 8 above, making sure you use the proper `/dev` device.

However, if you are on a server and `/pub` has been damaged, you must restore it before booting single-user, because not all of the utilities you need will be present. As above for 'root,' you start by loading and booting the mini-UNIX file system — step 2 above. The procedure is the same as for the 'root' file system, except that in step 4 you must create the file system with `/etc/mkfs` instead of `/etc/newfs` (`newfs` would allocate all of the 'g' partition to `/pub`, destroying the contents of the client `ndl#` partitions). The syntax for `/etc/mkfs` is explained in the section immediately above, *Restoring An Entire File System*; see step 1 there.

Once you have restored `/pub`, you need to install a new boot block for clients to boot from; use the commands:

```
# cd /usr/mdec
# installboot bootnd /dev/rxy0g
```

You can boot the system and proceed with the steps for reloading entire client partitions (that is, if any clients must be restored, for example if the whole `g` partition was wiped out). For restoring clients, follow the slightly different steps in the section above, *Restoring An Entire File System.*

Restoring Files Over Ethernet

You may restore files to a machine where you are logged in as superuser, from a tape mounted on a remote machine's tape drive. To make things work right, there must be an entry for the machine which you are logged into and restoring to, in the `/.rhosts` file on the machine with the tape drive. While superuser on the machine onto which you want to restore files, type:

```
# /etc/restore xf remote_machine:/dev/drive filename(s)
```

For 1/4 inch tape drives, remember to use the `b` flag, to duplicate the blocking factor given to the `b` flag of `dump` at dump time. There is no `c` flag for cartridge

tapes in `restore`.

For restores over the Ethernet, substitute the appropriate values for *remote_machine*, *drive*, and *filename(s)*. Of course, you may request an option other than `x` (such as `i` or `t`), but you must use the `f` to specify the remote tape device. See `restore`(8) for more about options.

3.5. Maintaining File Systems

Disk Capacity — Checking The Disk Resource Usage

Several commands will tell you about the current use of disk resources. Three commands will tell you the size of files in a file system, `ls`(1), `du`(1) and `quot`(8).

- `ls` gives you the size in kilobytes of the files in a directory when you pass the `-s` option, and recursively lists subdirectories with the `-R` option. To find the largest files in the current working directory, type:

  ```
  % ls -s | sort -nr | more
  ```

  To find out which files were most recently written, use `ls -t`. It lists files in order of most recently created, or altered, first.

- `du` will give the number of kilobytes contained in all files and, recursively, directories within each specified directory or file named. Note that `du` follows the symbolic links in a file system.

- `quot` must be executed by the superuser, and gives the number of blocks currently owned by each user in the named file system.

These three commands allow you to quickly find large files. If disk storage space is short, you can move unused large files to a less expensive medium like tape.

`df` — Show Free Space

The `df`(1) command returns information about specific disk partitions, including the file space occupied by each file system, the space used and the space available, and how much of the file system's total capacity has been used. When `df` gives the percentage of the capacity used on the disk, the number indicates the optimum capacity. When a file system is 90% full, `df` claims that the file system is 100% full. If the file system becomes any fuller, file system access slows down. In fact, UNIX permits file systems to become only 90% full unless you are the superuser. Ordinary users get an error if they go beyond this limit.

Please note that `df` will only return values for the first soft partition when a hard partition has been divided into a number of soft partitions. For example, only `/pub` shows up on an `nd` server's `xy0g` even though that disk also contains clients' soft partitions. This happens because `df` expects to read a file system, but reads instead the header for the first soft partition. Even though the header defines a file system smaller than the hard partition, `df` only returns values for the first soft partition.

Finally, there are two ways to find out how the disk is divided into hard and soft partitions. The `verify` command of `diag`(8) gives information about the size of the hard partitions on the disk. Run `dkinfo`(8) to obtain the same information without using `diag`. See the description of `diag` for more information. In addition, `/etc/nd.local` shows how a hard partition is divided into soft partitions, and the amount of space allocated to each soft partition. See `nd`(8) for a description of the contents of `/etc/nd.local`.

The Disk Quota System

Disk space is always a limited resource. The disk quota system provides a mechanism to control usage of disk space by users. Quotas may be set for each individual user on any or all file systems. The quota system warns users when they exceed their allotted limit, but allows some extra space for current work. A user who remains over quota longer than a specified time-limit will get a fatal over-quota condition.

The quota system is an optional part of the kernel, VMUNIX, that may be included when the system is configured.

Setting Up The Quota System

The system administrator must perform several steps to set up the disk quota system.

First, login on the NFS server machine; there the system must be configured to include the disk quota sub-system. Find the system configuration file in the `/usr/sys/conf` directory and, if it isn't already there, add the line:

```
options    QUOTA
```

After that, run `config`(8) and rebuild `vmunix` — for all the details of this process see the manual *Installing UNIX On The Sun Workstation.*

Second, decide what file systems should have quotas imposed. Usually, only file systems containing users' home directories, or other user files, will need quotas. However, you may want to include `/usr`. If possible, `/tmp` should be free of quotas. Work out your plan before you begin the actual allocation. After you decide which file systems should have quota restrictions, allocate available space according to your plan.

The `edquota`(8) command does the actual job of setting quotas for specified users. `edquota` is a quota editor that allows you to set limits for one or more users at a time. The **-p** option allows you to duplicate the quotas of a prototypical user for the a list of actual users. This is helpful when many or all users are given the same limits. The **-t** option edits the over-quota time-limit for the specified users. You may impose any combination of hard and soft limits for each user. A limit set to zero is disabled; if all the limits for a user are disabled, the entire quota for that user is also disabled and the system will not keep track of her usage. For details see the man page `edquota`(8).

Once quotas are set and ready to operate, they must be turned on to start working. `quotaon`(8) tells the system to start enforcing the quotas set with `edquota`. `quotaon` enables disk quotas for one or more file systems; the specified file systems must be mounted at the time. Quotas can be turned on at boot time by including the line:

```
/usr/etc/quotaon  -a
```

in the file `/etc/rc.local` on the NFS server machine. When that line is present all file systems in the server's `/etc/fstab` marked read-write with quotas will have their quotas turned on; for example:

```
/dev/xy2g /usr/labserver 4.2 rw,quota 1 3
```

Typically, the quota system is invoked at boot time as illustrated here. See `quotaon`(8) and `fstab`(5) for details.

The `quotaoff`(8) command will disable quotas — see man page for `quotaon`(8). However, when a file system is about to be unmounted by `umount`(8), the `umount`(2) system call disables quotas just before the unmount of the specified file system; this guarantees that the quota system will not be disabled if the unmount fails because the file system is busy.

Note that while the disk quota system is typically run on an NFS server, as shown in the examples above, you may also set up quotas on a standalone or timesharing machine.

Some Implementation Detail

Disk quota usage and limits are stored in the file `quotas` on the root of the file system where the quotas are imposed. This name is not known to the system in any way. However, several user level utilities depend on it, so choosing any other name would cause problems.

The data in `quotas` is an array of structures, indexed by `uid`, with one structure for each user on the system (whether the user has a quota on this file system or not). If the `uid` space is sparse, the file may contain holes which would be lost by copying. Therefore, avoid copying the file.

The `quotactl`(2) system call informs the system of the existence of the `quotas` file. Each subsequent open of a file on the file system will be accompanied by a pairing with its quota information. In most cases this information will be retained in core, either because the user who owns the file is running some process, or because other files owned by the same user are open, or because some file was recently accessed. If the information is not in memory, the quota file is read to retrieve it. In memory, the quota information is kept hashed by `uid` and file system, and retained in an LRU chain so that recently released data can be easily reclaimed.

Each time a block is allocated or released, and each time an inode is allocated or freed, the quota system gets told about it. The quota system gets an opportunity to object to any allocation.

The quota code uses a very small percentage of the system cpu time consumed in writing a new block to disk.

Administration Of The Quota System

All quota administration commands must be run on mounted file systems. The commands will work whether or not the quota system is disabled.

The Sun disk quota system is based on the 4.2BSD quota system. Sun's quota system limits the amount of time a user can be over quota. That time-limit may be adjusted on a per-file-system basis using `edquota`(8).

Periodically (certainly after each reboot, and when quotas are first enabled for a file system), the records retained in the quota file should be checked for consistency with the actual number of blocks and files allocated to a user. Do this using `quotacheck`(8); it is typically run from `/etc/rc.local` before `quotaon` — see the man page for details.

The super-user may use `quota`(1) to examine the usage and quotas for any user, and `repquota`(8) to check the usages and limits for all users on a file system.

The system administrator can increase or decrease a user's quota limits at any time. Thus, a user who exceeds his limits can ask the system administrator to increase them.

When you remove a user from a system, you should use `edquota`(8) to set the removed user's limits to zero. Then, when the user's login is removed there will not be an entry for it in the `quotas` file.

The User's View Of Disk Quotas

The `quota`(1) command provides information to a user about disk quotas when the system administrator has imposed quotas for that user; it also reports on current usage by a user. Two kinds of limitation may be imposed on a user; a quota can be set for the amount of disk space a user can fill up, or there may be a limit to the number of files (inodes) he can own. Typically, both limitations are imposed.

In the disk quota system, there are soft limits and hard limits. The soft limit is the number of 1K blocks (or files) that the user is expected to remain below. Each time the user's usage goes past this limit, he is warned, and a time limit is set. If the user remains above quota and the time limit expires, the particular limit that has been exceeded will be treated as if the hard limit has been reached, and no more resources will be allocated to the user. The **only** way to reset this condition is to reduce usage below quota. The system administrator may set the over quota time limits for each file system restricted by quotas.

The hard limit cannot be exceeded. If usage reaches this number, further requests for space, or attempts to create a file, will fail with an `EDQUOT` error. The first time this occurs, a message will be written to the user's terminal. Only one message will be written each time the hard limit is reached. Space occupied must be reduced below the limit.

What To Do When Quota Limit Is Reached

In most cases, to recover from exceeding your hard quota limit you must abort the process or processes in progress on the file system that has reached its limit, remove enough files to bring the limit back below quota, and retry the failed program.

The case is different when you are in the editor and a write fails because it exceeds your quota. Most likely, the write attempt which generated the quota exceeded message also truncated the file in the editor. If you abort the editor in these circumstances, you will not only lose recent changes, but probably some, or all, of the file proper as well. There are several safe exits from this situation. You may use the editor shell escape command (`!`) to examine your file space and remove surplus files. For example, you type '`!`' while in the editor and execute a command to the shell prompt that the system gives. However, it is probably quicker to use `csh` to suspend the editor session while you remove some files. For example, while in the editor type '**`CTRL-Z`**' (the CTRL key and the Z key simultaneously) and you will be returned to the `csh` for as many commands as you need to remove files and get below your quota limit. When you are ready to continue the editor session, just type '**`fg`**' (for foreground) to the shell prompt, and the suspended editor will restart. A third possibility is to write the file to some other file system (perhaps to a file on `/tmp`) where your quota has not been exceeded. After rectifying the quota problem, the file can be moved back to the

file system where it belongs.

Quotas And The NFS

Quotas on remotely mounted file systems work much the same as quotas on locally mounted file systems. The important difference is that no warnings are printed when the user goes over the soft limit. NFS clients should use `quota`(1) to find out the state of their quota allocation. Hard limit errors are treated like other hard errors in the NFS — for example exceeding actual disk capacity. When a user exceeds the hard limit she will get return code errors, not system error messages. The user may see `EDQUOT` error return codes from the system calls `write`(2), `fsync`(2) and `close`(2). Check these return codes to avoid disaster.

Making Room On File Systems

If you begin running out of room on a file system you need to free up some of the disk's storage space. If you get a message `filesystem full` on the system console of a running system, you should suspend or kill the currently executing programs and try to make more room on the affected file system. Of course it is better to monitor disk usage and not be caught in this situation, but that is not always possible. For systems that are filling up, or that become full suddenly, you will need to delete some files or move some files to another file system. These hints that may be useful.

1) Read the section above on *Disk Capacity*, then determine the largest files, their locations, and who owns them.

2) When UNIX crashes, it saves a core dump (roughly equal in size to the amount of physical main memory on your system) in a portion of the swap area on the disk. At the next reboot, `savecore`(8), which runs from `/etc/rc.local`, copies this core dump into files in the directory `/usr/crash`. In the `nd` server-client environment, core dumps are put in the directory `/usr/`*servername*`/crash/`*hostname* where *servername* is the name of the `nd` server, and *hostname* is the name of the machine the core dump is from. If you have a series of system crashes, then `/usr/crash` will keep accumulating core dump files until the file system fills up. The affected file system will typically be the `usr` partition (xy0g or sd0g) on standalone systems or the `/usr/`*servername* partition on `nd` servers and diskless clients.

   As distributed, the section of `/etc/rc.local` which creates the core dump file is commented out. Should you wish to get the core dumps without using up too much disk space, you can create a file called `minfree` in `/usr/crash`. `savecore` reads from this file, and if the file system contains less free space in kilobytes than the ASCII number contained in `minfree`, the core file will not be saved.

3) The directory `/usr/adm` contains accounting information. (Note that in the `nd` server-client environment, `/usr/adm` is a symbolic link to `/private/usr/adm`.) Check to see if any of those files are growing inordinately large. In particular, `/usr/adm/lastlog` can become enormous if large user id numbers (> 32K) are used in `/etc/passwd`. You are advised to avoid very large user numbers in `/etc/passwd`. You can disable login accounting by removing the file `/usr/adm/wtmp`. You can

disable process execution accounting by removing the file `/usr/adm/acct`. Re-enable accounting by recreating the file in question with length zero. For example, type: **`touch`** *filename*.

4 You can look for obsolete files in `/tmp` and `/usr/tmp`. Note that in the `nd` server-client environment, these are symbolic links to `/private/tmp` and `/private/usr/tmp`. Be careful not to delete currently active temporary files from `/tmp`, such as file names starting with `Ex` or `Rx`, which the editors use. Also look for obsolete files in the subdirectories of `/usr/spool`, such as `/usr/spool/mail` and `/usr/spool/uucppublic`. Note that in the `nd` server-client environment, `/usr/spool`, is a symbolic link to `/private/usr/spool`.

5) If you want to move a file or directory but programs expect to find it in a particular place, you can make a symbolic link. For example, if you want to move the 'system' directory `/usr/sys` onto a spare disk partition mounted on `/usr2`, become superuser and do the following:

```
# mkdir /usr2/sys
# chmod 777 /usr2/sys
# cd /
# mv sys sys.bak
# ln -s /usr2/sys
# mv sys.bak/* sys
# rmdir sys.bak
```

Warning: some system files, such as `vmunix` and `/dev`, should never be symbolically linked. One exception, on diskless clients only, `vmunix` is a symbolic link to `/pub/vmunix`; `boot` expects to find it there on diskless clients. In general however, moving a file from its expected place can cause big problems. If you are not sure whether a file can be safely moved, do not move it.

Files Needing Periodic Attention

See the chapter in this manual called *Periodic Maintenance* for a list of files needing periodic attention. The ones that should concern you regarding the allocation of disk space have been mentioned above in this section.

4

Communications

Communications **101**

4.1. The Ethernet 102
Ethernet Hardware 102
Example Of A Network Transfer 103
Configuring The Network In System Software 104
4.2. Wide Area Networks 117
An Overview Of `uucp` 117
Installing `uucp` 119
4.3. Setting Up The Mail Routing System 123
Picking a Name for your Domain 124
Picking a 'Main Machine' for Mail Forwarding 125
Testing Your Mailer Configuration 129
Diagnosing Troubles with Mail Delivery 129
4.4. Tip 131

4

Communications

This chapter discusses communications between machines. It begins with the two types of communications networks that your workstation can be connected to.

The first is a high-speed local area network (LAN), the Ethernet; the Ethernet allows machines in close proximity to share system resources such as disks and tapes. The chapter covers Ethernet terminology for both hardware and software, and explains Ethernet installation and debugging.

The second network type is a lower-speed wide area network that works over phone lines and allows distant machines to communicate with one another. Two aspects of wide area networks are introduced and explained in this section: `uucp`, a series of programs that allows UNIX systems to communicate with each other over dial-up or hardwired lines.

Next, a section covers the installation of `sendmail`, the electronic mail routing program for the UNIX system.

Finally we explain `tip`, which allows you connect to a remote machine as if you were logged in directly.

4.1. The Ethernet

Ethernet Hardware

The Ethernet is used for inter-computer communications in a limited area — within a building or between neighboring buildings. Each site can taylor the Ethernet to share workstation resources according to its needs. In this section we give a brief explanation of the hardware that comprises a basic network, and introduce terms and concepts that Sun users need to know.

Individual workstations and other devices are plugged into the Ethernet's cable system. Within workstations, each Ethernet board is assigned a unique address; therefore, each one can be moved around and plugged into any convenient outlet in the system. All devices plugged into the Ethernet can communicate with one another.

The diagram on the following page shows a typical Ethernet setup for a local network. Any local network may be connected to other local networks through gateway machines as explained below in the section *Setting Up A Gateway Machine.*

The basic parts of a typical Ethernet system, as shown in the diagram, include the following:

1) The Ethernet controller board in the workstation. controller).

2) The transceiver cable that connects the controller board to the transceiver tap on the coaxial cable. It is a 15 meter shielded, twisted pair cable that has four pairs, one each for transmit, receive, collision detect, and power. It has a 15 pin male connector on the controller end and a 15 pin female connector on the transceiver end. Transceiver cables can be joined together up to a maximum of 50 meters. Optionally, the transceiver cable can be connected to an Ethernet multiplexor box instead of to the transceiver on the coaxial cable.

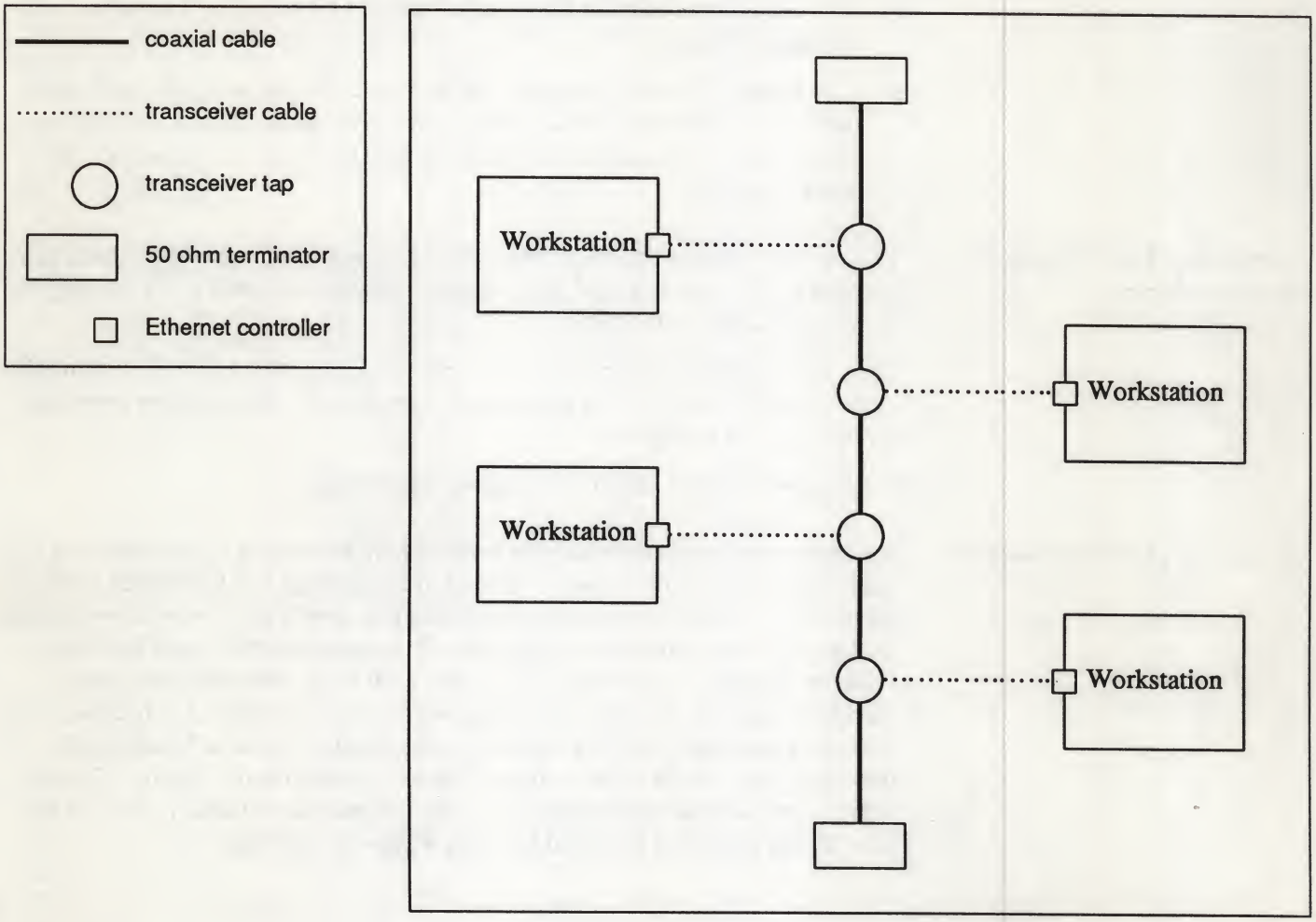
3) The transceiver tap into the coaxial cable. It makes a high impedance connection to the common coaxial cable and provides electrical isolation between the coaxial cable and the twisted pair cable. The transceiver passes signals to the coaxial cable from the controller. It also receives signals from the coaxial cable which appear on the receive lead of the transceiver cable. The transceiver taps must be placed on the coaxial cable at multiples of 2.5 meters in order to obtain the clearest signal.

4) An optional Ethernet Multiplexor box. A multiplexor box is equivalent to a number of transceivers stacked end to end and connected to the end of a transceiver cable. Up to 8 transceiver cables can be run from a single multiplexor box. The multiplexor box preserves signal clarity in the cables and allows you to attach several devices to a single transceiver tap into your coaxial cable.

5) The coaxial cable that forms the backbone of the Ethernet system. All workstations or other devices are attached to it with transceiver cable and taps. The coaxial cable is a 50 ohm cable with multiple shields to minimize susceptibility to strong RF fields. The maximum operative length of joined coaxial cables in an Ethernet system is 1500 meters.

6) Both ends of a coaxial cable are terminated by 50 ohm terminators with insulated outside covers.

Figure 4-1 *A Local Network*



Example Of A Network Transfer

Here is a breakdown of file transfer over the Ethernet.

1) A workstation user runs a file transfer program and specifies a file to be transferred between a sending and receiving device.
2) Software maps the file's characters into device-independent virtual characters to comply with protocol specifications.
3) The mapped character stream is routed to a virtual circuit, set up between the two devices.
4) The virtual circuit software breaks the character stream into packets for transmission.
5) The packets are passed to the Ethernet driver software.

6) The Ethernet driver copies the packet into a packet buffer and tells the controller to transmit it.

7) The controller waits until the coaxial cable is not in use and then transmits the packet.

8) The Ethernet transceiver receives the packet's bit stream and injects it into the coaxial cable.

9) The receiving station recognizes its address in the packets, and reverses the above procedure: bits are received by the transceiver, fed to the controller, passed to software that reassembles the packets, maps the characters, and stores the data.

Configuring The Network In System Software

This section explains the installation of network software, including setting up a gateway machine on the local net, reducing network overhead, and protecting the security of clients on the net.

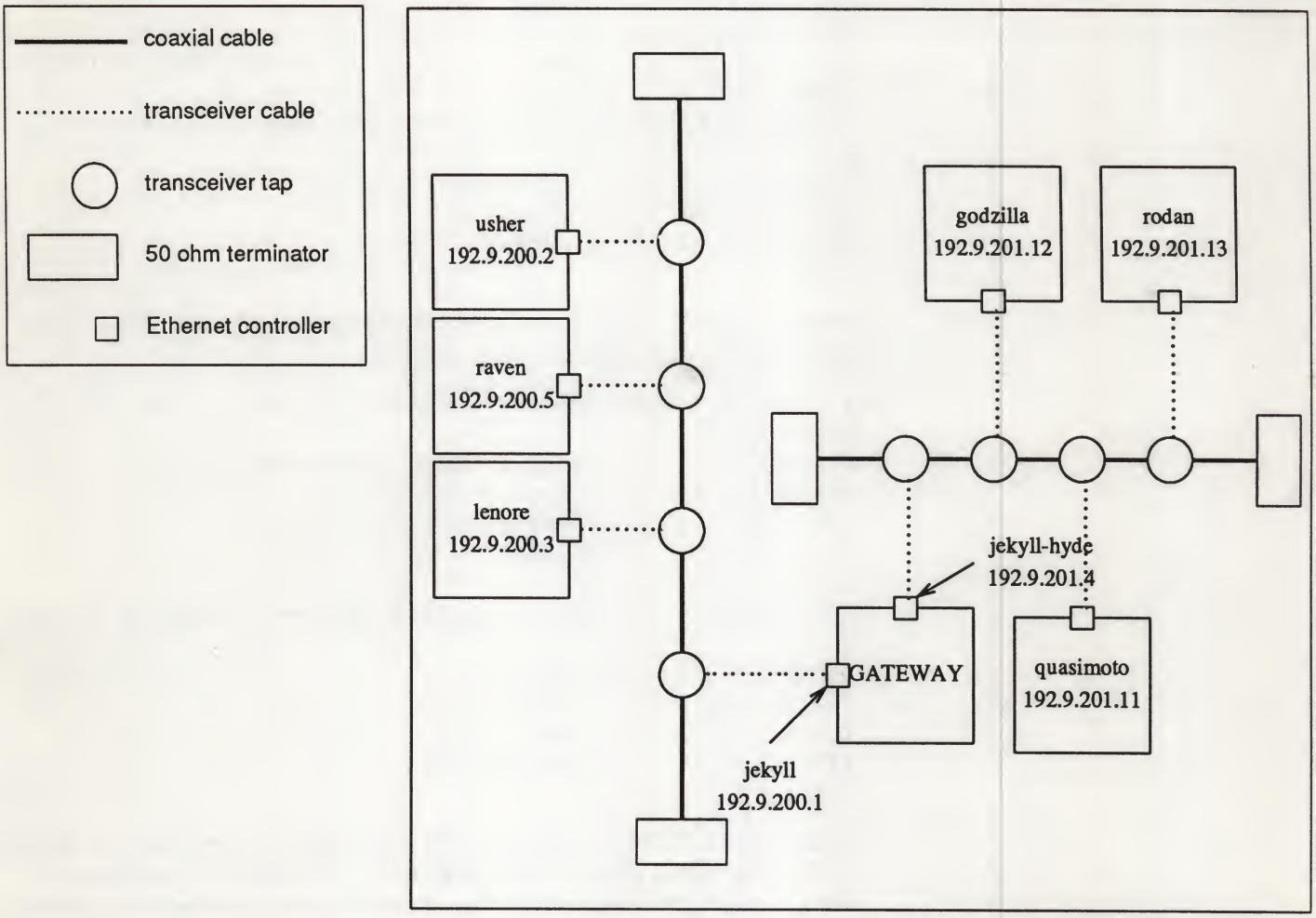
There is a short description of special cursor characters that occur during an Ethernet boot, followed by an explanation of how to make older networks compatible with current networks.

Finally, we list some hints for debugging the Ethernet.

Setting Up A Gateway Machine

A gateway machine joins two local networks. If you have a standalone local network, you don't need a gateway. If you have more than one terminated coaxial cable in your complete network system, you will need a gateway machine to join the local networks on each coaxial cable. A gateway machine must have two Ethernet Boards, one to communicate with each local network. Since each machine on a local network must have a unique hostname for each Internet address, a gateway machine must have 'two identities;' it must be assigned a hostname and Internet address for each local network it is on. Figure 4-2 below depicts two local networks joined by a gateway machine. Look it over and follow the steps below if you need to create a gateway machine.

Figure 4-2 *Two Local Networks Joined By A Gateway Machine*



To set up a proper configuration for a gateway machine you must edit `/etc/hosts` on the `yp` master server and then do a `ypmake` so that all yellow pages clients will see the change. You must also edit `/etc/rc.boot` on the gateway machine. Follow the four steps below.

1. On the `yp` master server machine edit the `/etc/hosts` database. You need to add a line for the gateway machine's hostname and address on the 'second' (or other) local network. For example, say 'jekyll' will be a gateway machine between two local networks. Remember, it must have two Ethernet interfaces. Before the change to gateway status, several lines of the `/etc/hosts` database on the yellow pages master server might look like this:

```
# Local Net 192.9.200 -- 10Mb/s Ethernet -- Engineering
#
192.9.200.1       jekyll loghost datehost
192.9.200.2       usher
192.9.200.3       lenore
192.9.200.5       raven
[ etc. ]
# Local Net 192.9.201 -- 10Mb/s Ethernet -- Marketeering
#
192.9.201.11       quasimoto
192.9.201.12       godzilla
192.9.201.13       rodan
[ etc. ]
```

To make 'jekyll' a gateway to the Marketeering local network, 192.9.201, add an entry for it's address on that network:

```
# Local Net 192.9.200 -- 10Mb/s Ethernet -- Engineering
#
192.9.200.1       jekyll loghost datehost
192.9.200.2       usher
192.9.200.3       lenore
192.9.200.5       raven
[ etc. ]
# Local Net 192.9.201 -- 10Mb/s Ethernet -- Marketeering
#
192.9.201.11      quasimoto
192.9.201.12      godzilla
192.9.201.13      rodan
192.9.201.4       jekyll-hyde
[ etc. ]
```

Notice that on the second network 'jekyll' has an 'alter-ego' name of 'jekyll-hyde'. The primary name here is 'jekyll', but for the network software to work properly, there must be a unique hostname for the gateway machine in the `/etc/hosts` database for each network. For ease of administration, it makes sense to use similar hostnames for each network — people on both networks can address the machine by the primary name (here jekyll), while the system administrator can tell the difference between the two.

2. This note about Internet addresses applies only if your gateway machine is also an `nd` server machine. When you configure an `nd` server as a gateway, its host number (the last field of the full Internet address) must be unique on both local networks. In effect this means that both of a server-gateway's host numbers are ineligible for use by any other host on each network. Taking the example of the two networks shown above, we notice that between host-names `lenore` and `raven`, the host number '`4`' was not used in Engineering Network 192.9.200. Presume we know that no host on the rest of that network uses '`4`' as its host number, then we are safe in assigning `jekyll` the host number '`4`' on the Marketeering Network since it is not used there either. Similarly, we see that on the Marketeering Network host numbers begin at '`11`' and therefore the host number '`1`' (`jekyll`'s number on the Engineering Net) is also unique on both systems. Thus, `jekyll`'s host

numbers are unique on both networks, as they must be for an nd server that is also a gateway machine. Precautions, such as a comment in the /etc/hosts, database should be taken to make sure these unique host numbers are not inadvertently assigned in the future. For example:

```
# Local Net 192.9.200 -- 10Mb/s Ethernet -- Engineering
#
192.9.200.1       jekyll loghost datehost
192.9.200.2       usher
192.9.200.3       lenore
# host number 192.9.200.4 taken by server-gateway on
# Marketeering Local Net: DO NOT USE
192.9.200.5       raven
[ etc. ]
# Local Net 192.9.201 -- 10Mb/s Ethernet -- Marketeering
#
# host number 192.9.201.1 taken by server-gateway on
# Engineering Local Net: DO NOT USE
192.9.201.11      quasimoto
192.9.201.12      godzilla
192.9.201.13      rodan
192.9.201.4       jekyll-hyde
[ etc. ]
```

3. Finally, run ypmake on the new /etc/hosts database to propagate it to all machines. Make sure you are in the directory /etc/yp and type the following:

   ```
   # make hosts
   ```

   This will update the /etc/hosts database and yppush it to the slave databases on the network. For details see ypmake(8).

4. On the new gateway machine, edit /etc/rc.boot, and add an ifconfig line mapping the machine's second hostname and Ethernet Board. Taking our example above, here is the relevant line from the file prior to addition:

   ```
   /etc/ifconfig xx0 jekyll -trailers up
   ```

   and the way it looks after the addition:

   ```
   /etc/ifconfig xx0 jekyll -trailers up
   /etc/ifconfig xx1 jekyll-hyde -trailers up
   ```

   For *xx* above, you substitute the proper Ethernet controller type, either ec for a 3COM controller or ie for a Sun-2 controller.

Reducing Network Overhead

As you read the following section, keep in mind that we strongly discourage the use of rwhod(8C).

You can have most of the advantages of living in a network environment without running the network daemons — rwhod(8C) and routed(8C) — on every machine. For systems with only 1 MByte of local memory, we advise disabling the daemons, or running them only on specific machines, so the space consumed by them can be made available for other purposes. When running, routed and

`rwhod` are actively doing something many times a minute, and leaving many pages in memory almost all the time. On a 1-MByte system about 7% of the 600KB of available user memory gets consumed. Combined, both daemons use up about 14% of available user memory.

`rwhod` allows the programs `rwho`(1C) and `ruptime`(1C) to return status information about machine usage of hosts on the local network. Unless you really need this information, we suggest you do not run `rwhod`. By default, `rwhod` is not run. If you choose to run it, remove the comment mark (a '#' in column 1) from the following line in `/etc/rc`:

```
# /usr/etc/in.rwhod & echo -n 'rwhod'    >/dev/console
```

If you do run `rwhod`, it may cause error messages on `nd` client machines of the form:

```
nd: disk server not responding
nd: server ok
```

When this occurs, you may also find a high rate of output packet collisions on a client when you check network status with `netstat -i` (see `netstat`(8)). The packet collisions occur when clients attempt to access their partitions (using `nd`) while simultaneously trying to satisfy a request for information by the `rwho` daemon, `rwhod`. The resulting burst of packet collisions prevents the clients from accessing their server for a long enough period to cause the `disk server not responding` message. The problem isn't fatal, but can be annoying.

Normally, `routed`, the routing daemon, runs on each Sun Workstation, and maintains network routing information that enables your machine to pick the best path for sending packets to external networks. `routed` consumes a small amount of memory and CPU time running an algorithm to keep accurate information about the topology of the local network.

Whether or where you should run `routed` depends on your system configuration. In general, if you have more than 1 MByte of memory in your workstation then it probably isn't worth thinking about whether you should run `routed`, just run it. With limited memory, see which one of the following best applies to you, and follow through:

1. If there are no gateway machines in your local network then you don't need to run `routed`. You can disable the daemon by removing (or commenting out — insert a '#' before each line) the following three lines in your `/etc/rc.local` file:

   ```
   if [-f /etc/in.routed]; then
      /etc/in.routed & echo -n 'routed'    >/dev/console
   fi
   ```

2. If you have a gateway then you **must** run `routed` on it.

3. If you have only 1 MByte of memory **and** only one gateway in your local network, then you can run `routed` only on the gateway machine, and disable it on all other nodes. All machines on the local network can then

redirect packets going to another accessible network through this gateway machine. To do this, edit the `/etc/rc.local` file on all machines except the gateway (*gateway_name*, in the example). Find the line that says:

```
echo -n 'local daemons:'    >/dev/console
```

Insert the following two lines just before it:

```
/usr/etc/route add 0 gateway_name 1
echo '  /usr/etc/route add 0 gateway_name 1'>/dev/console
```

Then, find the following three lines and comment them out (insert a '#' before each line) or remove them:

```
if [ -f /etc/in.routed ]; then
    /etc/in.routed & echo -n ' routed'  >/dev/console
fi
```

4. Finally, if you have only 1 MByte of memory **and** you have more than one gateway in your local network **and** you are running diskless, you can use your nd server's routing tables. (Of course, the server must run `routed` in this case so that clients can use its routing tables.) Edit the `/etc/rc.local` file on all clients of the server machine (*server_name*, in the example), and add the following lines:

   ```
   /usr/etc/route add 0 server_name 1
   echo '  /usr/etc/route add 0 server_name 1'>/dev/console
   ```

   Then comment out or remove the daemon's lines:

   ```
   if [ -f /etc/in.routed ]; then
       /etc/in.routed & echo -n ' routed'  >/dev/console
   fi
   ```

   Since a diskless node cannot run when its server is down, this always works; however, it causes packets to be sent through the server, loading it down.

In all other cases you should run `routed`.

Network Security — `/etc/hosts.equiv` and `.rhosts`

Network security is implemented at two levels: first, at the machine or node level, and, second, at the individual user level. The `/etc/hosts.equiv` and `.rhosts` files, respectively, control access at these levels. Remember that with the yellow pages `hosts.equiv` and `passwd` files are looked at in the yellow pages database on the `yp` server. If a machine does have its own copy of either of these files, the local copy will be looked at instead of the `yp` database. The security-checking process goes like this:

- When a user initiates a remote process on another machine (`rlogin`, `rsh` or `rcp`, for example), the system first checks for an entry for this user in `/etc/passwd` on the remote machine. If no entry is found, the user will be denied access: if he is trying to `rlogin` to the machine, he will be prompted for a password and then get a `Login incorrect` message; if he is attempting an `rcp` or `rsh`, he will get the `Login incorrect` message.

- If an entry for the user is found in `/etc/passwd`, the system next checks for his machine's hostname in the other machine's `/etc/hosts.equiv` file. If the hostname is found, the user gains access.
- If no `/etc/hosts.equiv` entry is found, the system checks for a line with his hostname (and, optionally, username) in the `.rhosts` file in his home directory on the machine to which he is trying to gain access. If the entry is found, the user gains access.
- If no entry is found in either `/etc/hosts.equiv` or *~user_name*`/.rhosts`, but the user is in `/etc/passwd`, the user is allowed to `rlogin` to the machine after giving his password, but gets `Permission denied` messages when attempting remote processes like `rcp` or `rsh`.
- The single exception to this security scenario is the super-user: the system skips the second-level check (`/etc/hosts.equiv` is not checked), and goes directly to looking at `/.rhosts`.

So, if you want to allow access to your machine by all users on another specific machine, include an entry for each user in `/etc/passwd` and include the machine's hostname in your `/etc/hosts.equiv` file, or make sure these are included in the appropriate `yp` databases. For example, if my machine's hostname is `gaia`, and I want to allow anyone on host `kepler` to gain access to `gaia`, I simply edit my `/etc/hosts.equiv` file as follows. The file is just a list of hostnames, one per line:

```
huey
ganymede
krypton
```

Add `kepler`'s name to the list:

```
huey
ganymede
krypton
kepler
```

Now all users who can gain access to `kepler` can also freely `rlogin`(1) to `gaia` (without being asked for a password), and can `rcp`(1) from, and use `rsh`(1) on, `gaia`, provided they are in `gaia`'s `/etc/passwd` file.

If you want to allow access to some users on a particular machine but not all, do not put the machine's hostname in `/etc/hosts.equiv`. Instead, put it in the `.rhosts` file in each user's home directory on your machine (*~user_name*`/.rhosts`). Note that, to avoid some security problems, this file must be owned by either this user or root, and may not be a symbolic link. The `.rhosts` file has a slightly different format than `/etc/hosts.equiv`: `/etc/hosts.equiv` accepts only hostnames; `.rhosts` accepts a hostname and, optionally, a user name on each line. Its format is best illustrated by an example. I can allow user `donald` at host `canard` to gain access to my machine, `gaia`, and keep other users of `canard` out by (1) making sure `canard` is not in my `/etc/hosts.equiv` file, and (2) adding an entry for `donald` at `canard` to `~donald/.rhosts`. The entry looks like this:

```
canard donald
```

Note also, that this means `donald` only has access when he's coming from `canard`; if he tries to use `gaia` from another host, he must know his password to be able to `rlogin` and can't complete remote processes. Of course, if he can `rlogin` he can always doctor his `.rhosts` file (if it is not owned by root, and I have given him a home directory). This brings up two points:

- The only way to achieve anything resembling security in a hostile environment is to exclude users from the `/etc/passwd` file; once someone knows a password, he's in. If this concerns you, you should also be especially careful to protect `/.rhosts` properly — make sure only root has write permission. Clearly, tight security has not really been an issue in the administration of previous UNIX systems; the goal has been to facilitate, rather than deny, access. See the section on *Yellow Pages Administration* for more information about how to edit the password file to exclude users when you run `yp`.
- If you have a friendlier environment, the easiest way to administer machine access at the user level is to give each 'trusted' user an account (in `/etc/passwd`) and a home directory on your machine(s), and ask each user to create his/her own `.rhosts` file in his/her home directory on the machine(s).

Finally, a caution about editing the `/etc/hosts.equiv` and `.rhosts` files. An entry in either of these files is invalid if it contains trailing white space — blanks or tabs. The presence of trailing white space may not be obvious. You can look at a file with `cat -e`, or `vi` with the 'set list' option, both of which will show a `$` at the end of each line. You may also `grep` a file searching for 'tab_character$' or 'blank_space$' patterns to get a listing of any lines ending in tabs or blanks.

Special Cursor Characters During Ether Boot

When booting over the Ethernet, you will notice that the cursor character changes from its usual solid block form to some other character. For example:

—, =	The cursor alternates between '—' and '=' to indicate that the boot is proceeding normally.
`X`	An Ethernet packet-transmission error has occurred.
`?`	An Ethernet packet-receive error has occurred; or a packet has not been successfully received for two seconds; or the network or `nd` configuration files are incorrect.

None of these characters appears when booting from disk or tape.

The `X` or `?` characters are rare. You may see the `?` sometimes when booting a number of clients simultaneously on the same server. If the `X` or `?` appears frequently when booting, and you have ruled out errors in the configuration files, it probably indicates an Ethernet hardware malfunction which should be corrected.

How To Make Current Networks Compatible With Older Networks

The network addressing format changed in Release 1.1. It only concerns you if:

- you have a network of machines running version 1.0 (or lower) of Sun software, and you want them to communicate with your new workstation(s) running 1.1 or newer, or
- You have a machine that cannot perform ARP (an older VAX, for example), and you want it to talk to your 1.1, or newer, Sun's.

If neither of these is the case, don't worry about it.

To make machines running 1.1, or a subsequent release, talk to 1.0 networks or machines which do not respond to or perform ARP, you can do one of three things:

1. The best path to convert your 1.0 network to the Class C addressing scheme is described below. If this is impossible in your system — for example, if your machine cannot perform ARP, or if you have older 4.1c Vaxen that cannot perform ARP on host numbers equal to or greater than 1024 — you must follow the third path. Otherwise, we urge you to convert.

   Conversion is relatively easy. You'll need to assign your old network a new network number (use '192.200.1' if you wish); assign each machine on the old network a new, unique host number between 1 and 254. When the new numbers have been decided upon, log on to the yellow pages master server and edit the `/etc/hosts` file. Change your old network number to your new network number, and identify each host with its host number. For example, if the old `/etc/hosts` file on the `yp` master server looked like this:

   ```
   125.5143    alpha
   125.5204    beta
   125.6422    gamma
   125.0x5245  delta
   125.0x2226  epsilon
   ```

   The new one might look like this:

   ```
   192.9.200.1 alpha
   192.9.200.2 beta
   192.9.200.3 gamma
   192.9.200.4 delta
   192.9.200.5 epsilon
   ```

   In the example above, `192.9.200` is the network number, and `1` is alpha's host number.

   After editing, while still on the `yp` master server, do the following to update the yellow pages map and propagate that new map to all other `yp` server machines — note that the change will not show up inless you complete these two commands:

   ```
   # cd /etc/yp
   # make
   ```

   Finally, reboot every machine whose host number you changed.

2. A second solution is to retain your Class A addressing scheme from 1.0, but make sure all host numbers are smaller than 1024, so that ARP can be performed.

   Follow the `/etc/hosts` editing procedure outlined in step 1 above, and assign each machine a host number — the last component of the entire address — under 1024. You can simply retain your old network number. Remember to do the `cd` and `make` commands, and to reboot every machine whose host number you changed.

3. Finally, if you have machines that either cannot perform ARP (old Vaxen, for example), or cannot respond to ARP (older Sun systems with Class A networks and host numbers greater than 1024, for example), there is a solution for each problem.

   First, machines which cannot perform ARP may talk to a post 1.1 network by 'forcing' your post 1.1 machines to respond to an address which non-ARPers understand. Do this by adding an `ifconfig` line to the `/etc/rc.boot` file on each post 1.1 machine. The new line contains the post 1.1 machine's 6-byte hexadecimal Ethernet address:

   a) You need the Ethernet address. To get it, in most cases, power on the Sun Workstation and, as soon as the self-test completes, abort. Then read the Ethernet address from the PROM monitor's sign on messages. (Remember that the correct abort sequence depends on your keyboard, as described in the manual *Installing UNIX On The Sun Workstation.*)

      A few sites may have very old machines that require a different process. For machines with PROMs of Rev. L or earlier, or for machines without CPU ID PROMs, you obtain the Ethernet address as follows. For a 3COM Ethernet Board use the PROM Monitor to interrogate the Multibus memory-space where the address is stored. As super-user, run the following sequence of commands (type <RETURN> where indicated):

```
# /etc/fasthalt
syncing disks . . . done
Unix Halted
> k1
> e fe0400
> FE0400:   0260? <RETURN>
> FE0402:   8C00? <RETURN>
> FE0404:   9920? q
```

      Note, if your machines have Sun-2 Ethernet Boards, this simply won't work. Instead, you can fabricate 3COM addresses for these machines: any six-byte hexadecimal number will do, as long as it is **unique** on your network.

   b) When you have the Ethernet address for a post 1.1 machine, edit the `/etc/rc.boot` file and add the following line for every pre 1.1 machine you want to communicate with:

```
/etc/ifconfig e_intface# ethernet_address `hostname` -trailers up
```

Use the correct device abbreviation for *e_intface* — `ec` for a 3COM Ethernet Controller, or `ie` for a Sun-2 Ethernet Controller — and the correct unit number of the controller for # — `0` for the first Ethernet board, `1` for the second. For *ethernet_address*, substitute the machine's entire 6-byte hexadecimal Ethernet address. For instance, the lines for the machine used in the example above might look like this:

```
/etc/ifconfig ec0 `hostname` -trailers up
/etc/ifconfig ie0 `hostname` -trailers up
/etc/ifconfig ec0 2:60:8C:0:99:20 `hostname` -trailers up
```

Remember to add a line for each pre 1.1 machine you want to communicate with.

c) Finally, reboot every machine whose `/etc/rc.boot` file you altered in this way.

Now, for machines that will not respond to ARP, you can use the `/etc/arp` command to specify the Ethernet addresses for the machines (see `arp`(8)).

Ethernet Troubleshooting

From time to time, most networks will have problems. Always check your network connections first. On networks such as the Ethernet a loose cable tap or misplaced transceiver cable can result in severely deteriorated service. The `netstat`(8) program may assist in tracking down hardware malfunctions. In particular, look at the `-i` and `-s` options on the manual page.

If you believe you have a faulty coaxial Ethernet cable, test it to make sure it delivers 50 ohms — see *Hardware Checks* below.

If you suspect a routing daemon malfunction, you may log its actions — and even all the packet transfers. To create a log file of routing daemon actions, just supply a file name when you start up the daemon, for example:

```
# /etc/routed /etc/routerlog
```

Whenever a route is added, deleted, or modified, a log of the action and a history of the previous packets sent and received will be printed in the log file. To force full packet tracing, specify the `-t` option when the daemon is started up. Beware though — on a busy network this generates almost constant output.

Sometimes, even after carefully hooking up your machines, you will have a problem starting up your network and will get a message like: `No server -- giving up` when trying to boot a diskless client, or `Connection timed out` when trying to `rlogin` between machines.

Below are some suggested corrective actions. After each step, the net should be tested again.

Software Checks

1) Check `/etc/hosts` on the `yp` master server machine to make sure that the entries are correct and up to date. Make sure a correct copy has been sent to all `yp` slave servers. Check the `ether` addresses in `/etc/ethers` on the `yp` master server machine to make sure that the entries are correct and up to date. Make sure a correct copy has been sent to all `yp` slave servers. Note, different rules apply to pre 1.1 Sun software releases.

2) If you cannot boot a client, you try specifying the host number of the server in the boot command. The syntax is:

```
> b xx (0,#,0)vmunix
```

Where *xx* is either `ec` for a 3COM Ethernet controller, or `ie` for the Sun-2 Ethernet controller; and where # is the host number. If this works, while the default boot does not, the server files are not configured correctly. Check `/etc/hosts` on the `yp` master server machine and/or `/etc/nd.local`.

3) Check the accuracy of the `/etc/ifconfig` line(s) added to the `/etc/rc.boot` file.

4) On a standalone host or a server, try `rlogin` to yourself. Make sure the network daemon `inetd` is running on the machines that want to talk to each other.

5) Run `/etc/nd` 'by hand' to produce error messages if there are errors. For example,

```
# /etc/nd < /etc/nd.local
```

5) `netstat`(8), with the `-i` option, tells you how many packets a machine thinks it is transmitting and receiving on each local network. For example, on a server you may see the input packet count increasing each time a client tries to boot, while the output packet count remains steady. This suggests that the server is seeing the boot request packets from the client, but does not realize it is supposed to respond to them. This might be caused by an incorrect address in `/etc/hosts` or `/etc/nd.local`. On the other hand, if the input packet count is steady, the machine does not see the packets at all, which suggests a different type of failure, possibly a hardware problem.

Hardware Checks

1) With its host system powered on, after awhile, each transceiver should feel slightly warm to the touch. A cold transceiver probably is not receiving power. This could indicate one of several things: a loose connection on either end of the transceiver cable, a loose connection of the internal Ethernet cable to the Ethernet board, or a faulty cable, transceiver (less likely), or Ethernet board (even less likely).

2) Check the solidity of all Ethernet coaxial and transceiver cable connections. If boards were changed or moved in the workstation card cage, make sure that the internal connector (from the back panel) is plugged into the Ethernet

board.

3) Is the network terminated on both ends? It should be. Unscrew one of the terminators and use an ohm meter to test resistance across the coaxial connector where you just unscrewed the terminator (use the pin 'inside' the N-connector for signal, and the housing for ground). You should measure about 50 ohms. If you get something other than 50 ohms, your cable may be damaged. This check is particularly pertinent if you use a clamp-on ('vampire clamp') type of transceiver; they tend to short-circuit the Ethernet coaxial cable. The threaded connectors on the type of transceivers supplied by Sun can also develop shorts at the connectors.

4) Remove one of the terminators and try operating the network. You should get error messages on every machine, something like: `Ethernet transmission error`. If a machine just continues to give its previous error (`Connection timed out` or `no server`) then it may not be connected correctly to its transceiver. As in check 3 above, the problem could be loose connections or a faulty connector, cable, transceiver, or Ethernet board.

5) Try swapping transceivers and/or transceiver cables. If spare Ethernet boards are available, try swapping boards. Even if there are only two machines on the net, exchanging parts may be informative. When swapping Ethernet boards, remember to edit the Ethernet addresses in `/etc/nd.local` where applicable.

4.2. Wide Area Networks

An Overview Of uucp

uucp (UNIX to UNIX copy) is a series of programs designed for communication, via dial-up or hardwired lines, between two systems running UNIX. uucp may be used to transfer files between UNIX systems, and also to run commands on remote machines. For more detailed background, see the *Uucp Implementation Description* in the *Tutorials* section of this manual.

Support for uucp is located in three major directories: /usr/bin (which contains user commands), /usr/lib/uucp (operational commands), and /usr/spool/uucp (spooling area).

(PLEASE NOTE: In the nd server — diskless client environment, the directories /usr/lib/uucp and /usr/spool/uucp are exported from the nd server. Since all machines share these file systems, machine specific files cannot reside in them. Instead the directories /private/usr/lib/uucp and /private/usr/spool/uucp contain the files that would normally reside in /usr/lib/uucp and /usr/spool/uucp. During installation setup replaces the files in them by symbolic links to the files in /private versions so the files can still be referenced by their normal names. If you are working in the nd server — client environment, remember to use instead the corresponding file in /private/usr/lib/uucp or /private/usr/spool/uucp when you follow the instructions below.)

The uucp commands are:

/bin/rmail	receive remote mail	rmail(8)
/usr/bin/rnews	receive remote news	
/usr/bin/uucp	file-copy command	uucp(1C)
/usr/bin/uux	remote execution command	uux(1C)
/usr/bin/uusend	binary file transfer using mail	uusend(1C)
/usr/bin/uuencode	uusend binary file encoder	uuencode(1C)
/usr/bin/uudecode	uusend binary file decoder	uudecode(1C)
/usr/bin/uulog	scans session log files	uucp(1C)

The important files and commands in /usr/lib/uucp are:

/usr/lib/uucp/L-devices	list of dialers and hardwired lines
/usr/lib/uucp/L-dialcodes	dialcode abbreviations
/usr/lib/uucp/L.cmds	commands remote sites may execute
/usr/lib/uucp/L.sys	systems to communicate with and how to connect
/usr/lib/uucp/SEQF	sequence numbering control file
/usr/lib/uucp/USERFILE	remote site pathname access specifications
/usr/lib/uucp/uucico	uucp protocol daemon
/usr/lib/uucp/uucp.day	script for daily polling/cleanup
/usr/lib/uucp/uucp.hour	script for hourly polling
/usr/lib/uucp/uucp.night	script for nightly polling
/usr/lib/uucp/uucp.noon	script for midday polling
/usr/lib/uucp/uucp.week	script for weekly cleanup of uucp log files
/usr/lib/uucp/uupoll	site-polling script
/usr/lib/uucp/uuclean	cleans up garbage files in spool area
/usr/lib/uucp/uuxqt	uucp remote execution server

The spooling area, /usr/spool/uucp, contains the following important files and directories:

/usr/spool/uucp/C.	directory for command, 'C.' files
/usr/spool/uucp/D.	directory for data, 'D.' files
/usr/spool/uucp/D.*hostname*	directory for local 'D.' files
/usr/spool/uucp/LOGFILE	log file of uucp activity
/usr/spool/uucp/SYSLOG	log file of uucp file transfers

Note that C., D., and D.*hostname* are subdirectories, unlike earlier implementations of uucp. In older versions, C. and D. files are placed directly into the spooling directory, /usr/spool/uucp; in the current version, they are placed in their appropriate subdirectory. So, in the old version you'd have, say, /usr/spool/uucp/C.res45n0031; but in the current version the file is /usr/spool/uucp/C./C.res45n0031.

As uucp operates it creates (and removes) many small files in the directories underneath /usr/spool/uucp. Sometimes files are left undeleted; these are most easily purged with the uuclean program. Instructions in the uucp.day file take care of doing this daily clean-up for you. The uucp log files can grow without bound unless trimmed back; maintain these files with uulog. uucp.day and uucp.week manage this housekeeping. If you decide to prune these directories yourself, be careful: randomly removing files from /usr/spool/uucp may cause uucp to generate error messages when it tries to access a file another file claims is there. (For instance, each mail transaction creates three files.) You do, however, need to clean the /usr/spool/uucppublic directory 'manually'; people at other sites send to it when sending files to users at your site.

uucp occasionally sends mail about minor problems to 'uucp' or 'root'. The system administrator should read and toss these messages. You can redirect these mail messages to your mailbox with an entry for 'uucp' in /usr/lib/aliases. Look at the examples there.

Under normal conditions, uucp calls your designated sites at specified times and, in addition, checks to see if any files are queued on that site's machine for you. If you ever need to invoke uucp 'on command', the line:

```
# /usr/lib/uucp/uucico -r1 -ssitename
```

forces uucp to poll *sitename*, even if there is nothing waiting. If you do run uucp in this fashion, don't run it as super-user, since the suid* bit will not be honored. If you have trouble with the connection, run uucp with the debugging option, as described in installation step 7, below.

Installing uucp

A uucp network link using modems or dedicated lines may be established between two machines running UNIX. To establish a connection between two sites that both have modems, one site must have (at least) an automatic call unit (an auto-dial modem) and the other must have (at least) a dialup port (an auto-answer modem). It is better if both sites have one of each or have modems which both call and answer, like Ventel's.

If both you and the site(s) you wish to connect with have autodial units and ports, install uucp as follows. If you have only a dialup your situation will be different; procedures are described near the end of this section. For more information, read the *Uucp Implementation Description* in the tutorials section of this manual; it describes in detail the file formats and conventions, and will give you necessary context.

- The name of the host machine in /etc/rc.boot will be the name of your uucp connection to the outside world (make sure it is fewer than 8 letters). We will call this machine your 'uucp host'; its name is your 'uucp hostname'.

  If this machine is your 'main machine' (for a definition see the next section, *Setting Up the Mail System*), then your uucp hostname should be the same as your domain name.

- Change the /usr/spool/uucp/D.noname directory to your own site's /usr/spool/uucp/D.*hostname* directory with the following:

  ```
  # mv /usr/spool/uucp/D.noname /usr/spool/uucp/D.hostname
  ```

  Use your uucp hostname for *hostname*.

- Create a uucp account for each remote host in the password file on your uucp host machine. Use /etc/vipw to enter a line of the following form in /etc/passwd:

  ```
  Uhostname:*:4:4::/usr/spool/uucp:/usr/lib/uucp/uucico
  ```

  Note, make this change only on the machine handling uucp traffic, not on the yp master server. Now use the passwd command to establish a password for each remote host:

* 'suid' stands for 'set user id;' if you set this bit on an executable file, UNIX will grant or deny file access based on the permissions of the file's owner, rather than the permissions of the person who executes the file. uucp uses this facility to ensure that all the files in its spool directories are readable and writable no matter who invokes the uucp program.

 # passwd U*hostname*

Tell this password to the administrator of the remote host, who must then incorporate it into the `L.sys` file on that machine.

- The `L.sys` file contains the phone numbers and login sequences required to establish a connection with a `uucp` daemon on another machine. Edit `/usr/lib/uucp/L.sys`, adding a line of the following form for each site you want to talk to:

  their_host **Any** *device baud phone#* **login:–EOT–login:uucp ssword:***password*

  The first field is the `uucp` hostname of the other site, the second indicates when their host may be called, the third field specifies how their host is connected (through an ACU, a hardwired line, etc.), then comes the baud rate of the line, phone number to use to connect through an auto-call unit, and finally a login sequence ending with a password. The phone number may contain common abbreviations which are defined in the `L-dialcodes` file. The device specification (third field) should refer to devices specified in the `L-devices` file. Note that Sun supports three currently manufactured modems: The Ventel 1200-31, the Ventel 1200-32, and the Hayes Smartmodem 1200. Sun will continue to support the Ventel MD 212-4. Indicating only `ACU` causes the `uucp` daemon, `uucico`, to search for any available auto-call unit in `L-devices`. For example, our `L.sys` file looks something like:

```
adiron Any ACUHAYES 1200 7620183 login:-EOT-login: uucp ssword: secret
ucbvax Any,20 ACUVENTEL 1200 6728212% login:-EOT-login: uucp ssword: almama
ucbarpa Any,20 ACUVENTEL 1200 4539351 login:-EOT-login: uucp ssword: netnut
decvax Any ACUVENTEL 1200 6139949241 login:-EOT-login: Ujedi ssword: cannon
```

For hardwired lines, your `L.sys` lines should contain the tty device name in the third and fifth fields:

```
anyname Any ttyb 1200 ttyb login:-EOT-login: uucp ssword: sunrise
```

- Connect your auto-dial/auto-answer modem to ttya (the port labeled 'SIO-A' on the backpanel of your Sun Workstation) on your host machine. There is a complete discussion of *Adding A Modem To Your System* in this manual in the chapter *Making Additions To Your System.*
- To set up your modem for both dial-in and dial-out on the same serial port, the flags bit corresponding to the serial port has to be set to zero in the kernel. To find out how to do this, read the *Kernel Modification* section of *Adding A Modem To Your System.*
- Create the appropriate device for your modem with the following series of commands:

```
# cd /dev
# mknod cua0 c 12 128
# chmod 600 cua0
# chown uucp cua0
# mv ttya ttyd0
```

- Edit `/etc/ttys` to include an entry for `ttyd0` (see `ttys`(4)). Insert the line: `13ttyd0`. Then type the following to initialize everything properly:

  ```
  # kill -1 1
  ```

- In some older releases, `uuxqt` may fail because it cannot open files due to permission problems. Check permissions on the following files (note that *systemname* below should be fewer than 7 characters):

  ```
  /usr/spool/uucp/C.
  /usr/spool/uucp/D.
  /usr/spool/uucp/D.systemname
  ```

  If they are not `rwx--x--x` (711), type the following:

  ```
  # chmod 711 /usr/spool/uucp/{C.,D.,D.systemname}
  ```

- Make sure your modem is hooked up properly by running `tip` with the phone number of a known machine:

  ```
  # tip 5551234
  ```

  If you get a `dialing . . . connected` response, all is well. If you get any other response, reconnect your modem.

- As a final test, run `uucp` with the debug option (`-x`), as follows:

  ```
  # /usr/lib/uucp/uucico -r1 -ssitename -x7
  ```

  With `-x`, the higher the number, the more debugging output you get; 1, 4, and 7 are reasonable choices. If you get substantial quantities of output from this command, everything is fine; you can go on to edit the files below.

- Edit the files `uucp.day`, `uucp.noon`, and `uucp.night`, in the `/usr/lib/uucp` directory. Each of these files has a 'for' statement which arranges to call sites you want to call at designated times for mail. In each file, find the line that says:

  ```
  for i in sys.name
  ```

  and change `sys.name` to the site name(s) you want to poll. For example:

  ```
  for i in ucbvax ucbarpa Shasta navajo
  ```

  will poll four sites.

- Add the `uucp.day`, `uucp.noon`, `uucp.night`, `uucp.hour`, and `uucp.week`, files to `/usr/lib/crontab` (see `cron`(8)). These arrange for the appropriate sites to be polled at the appropriate times. For example, the entries in `crontab` might look like:

```
5 6 * * * su uucp < /usr/lib/uucp/uucp.day
15 12 * * * su uucp < /usr/lib/uucp/uucp.noon
30 23 * * * su uucp < /usr/lib/uucp/uucp.night
10,30,50 * * * * su uucp < /usr/lib/uucp/uucp.hour
0 7 * * 2 su uucp < /usr/lib/uucp/uucp.week
```

If you have only a dialup, you can be a second-class citizen on the `uucp` net. You must find another site that has a dialer, and have it poll you regularly (once a day is a reasonable minimum). When you send mail to another site, you must wait for it to call you. To handle installation for a passive node, just complete steps one through five and step seven in the procedures above. When you come to the fourth step, editing `/usr/lib/uucp/L.sys`, you don't need to specify all the information called for in the step: only the first two fields of `L.sys` are necessary, and in practice only the first field (site name) is looked at. A typical `L.sys` for a passive node might be:

```
ucbvax      None
research    None
```

where the first field on each line is a site that will poll you. Next, put a password on the `uucp` login. Then let the other site know your phone number, `uucp` login name, and password.

4.3. Setting Up The Mail Routing System

UNIX allows you to implement a general internetwork mail routing facility called `sendmail`. `sendmail` features aliasing and forwarding, automatic routing to network gateways, and flexible configuration.

This section will tell you how to install the mail system on your system. For more detail there are two papers in the *Tutorials* section at the end of this manual: *Sendmail — An Internetwork Mail Router* and *Sendmail Installation And Operation Guide.*

The mail system consists of the following commands and files:

`/bin/mail`	old standard mail program (from V7)
`/usr/ucb/mail`	UCB mail program, described in `mail`(1)
`/usr/lib/sendmail`	mail routing program
`/usr/lib/sendmail.cf`	configuration file for mail routing
`/usr/lib/sendmail.fc`	"frozen" configuration file
`/usr/lib/sendmail.main.cf`	configuration file for 'main machines' (see below)
`/usr/lib/sendmail.subsidiary.cf`	configuration file for 'subsidiary machines' (see below)
`/usr/spool/mail`	mail spooling directory for received mail
`/usr/spool/mqueue`	spool directory for mail going out over the network
`/usr/spool/secretmail`	secure mail directory
`/usr/bin/xsend`	secure mail sender
`/usr/bin/xget`	secure mail receiver
`/usr/bin/enroll`	to receive secure mail messages
`/usr/lib/aliases`	mail forwarding information
`/usr/ucb/newaliases`	command to rebuild binary forwarding database
`/usr/ucb/biff`	mail notification enabler
`/usr/etc/in.comsat`	mail notification daemon
`/usr/etc/in.syslog`	error message logger, used by `sendmail`

Note for `nd` file server configurations:

In the file server and diskless client environment, the directory `/usr/lib` is exported from the server. Since this file system is shared by all the machines, machine specific files cannot reside here. Instead on servers and clients there is a directory called `/private/usr/lib` that contains the files that would normally reside in `/usr/lib`. The `setup` program replaces the files in `/usr/lib` by symbolic links to the files in `/private/usr/lib` so the files can still be referenced by their normal names. The files (and directories) so affected are listed below:

```
/usr/lib/sendmail.cf
/usr/lib/aliases
/usr/lib/aliases.dir
/usr/lib/aliases.pag
/usr/lib/crontab
/usr/lib/uucp
/usr/lib/news
```

When the following instructions tell you to copy or edit one of the above files, instead use the corresponding file in `/private/usr/lib`.

Note for diskful machines which mount the mail system from an NFS server:

On NFS servers and their clients `setup` does not automatically make the `/private/usr/lib` directory and the symbolic links as it does on an `nd` server and its clients. You must configure the NFS server and the diskful machine 'by hand' to resemble the `nd` server or `mail` will not work.

Mail is normally sent and received using `mail`(1), which provides a front-end to edit the messages sent and received, and passes the messages to `sendmail`(8) for routing. `mailtool`(8) is also available for this purpose. The routing algorithm uses knowledge of the network name syntax, aliasing and forwarding information, and network topology, as defined in the configuration file `/usr/lib/sendmail.cf`, to process each piece of mail. Local mail is delivered by giving it to the program `/bin/mail` which adds it to the mailboxes in the directory `/private/usr/spool/mail`, using a locking protocol to avoid problems with simultaneous updates. After the mail is delivered, the local mail delivery daemon `/usr/etc/in.comsat` notices, and it notifies users who have issued a `biff y` command that mail has arrived.

Mail for *username* is normally accessible in the file `/private/usr/spool/mail/`*username*. In the distribution system, your mail file is readable only by you; however, anyone with the super-user password can read others' files, including their mail. To send mail which is secure against any possible perusal (except by a code-breaker), use the secret mail facility, which encrypts the mail so that no one can read it. See `xsend`(1). Note that this facility does not work over the network.

The following subsections give some instruction on `sendmail` installation; for more detailed information, see the *Sendmail Installation and Operation Guide* in the tutorials section of this manual.

Picking a Name for your Domain

The network mail delivery program, `sendmail`, uses Internet standard mail addressing formats which make it possible for any Internet system in the world to send or receive mail with any other Internet system. To accomplish this, each system must have a unique name. To make it easier to assign names, the world of mail addresses is broken up into domains and subdomains.

For example, many systems funded by the U.S. Government's Advanced Research Projects Agency are in the domain 'arpa'. Many of the systems which send mail via the `uucp` protocols are in the domain 'uucp'. Within each domain, names have to be unique — there can't be two machines called 'MIT-Multics.arpa' or it would be impossible to decide how to deliver mail to them.

Domains nest inside one another like directories, except that the names go from right to left. Thus 'joe.sun.uucp' is host 'joe' in subdomain 'sun' which is in domain 'uucp' just like '/usr/lib/crontab' is file 'crontab' in subdirectory 'lib' in directory 'usr'. To make life easier for the people who maintain mailers, there are only a small number of 'top-level' domains like 'uucp' and 'arpa'.

If your workstation is on the Arpanet, or shares an Ethernet with a machine on the Arpanet, or with a machine on the Arpa Internet, you should probably pick a name in the 'arpa' domain, and register it with the Internet naming authority at

the Arpanet Network Information Center. Otherwise, pick a name in the 'uucp' domain.

If your workstation and/or Ethernet have no phone lines or connections to other networks, the name you pick within the 'uucp' domain doesn't matter much. You may, however, have to change the name if you get other network connections and somebody else is already using that name.

If your organization already has hosts on the `uucp` network or the Usenet, ask a system administrator for help in picking a name.

If you are connected to the `uucp` network, you must be talking with at least one other computer on the network. Check with the system administrator of that machine to see if the name you have picked is already in use in the `uucp` network. If they are on the Usenet, they can look in newsgroup 'net.map'; if not, they just have to guess.

In a network of Suns, the name of the 'main machine' becomes the name of your subdomain, and each other machine can truly have any name you want. For example, a main machine at a site might be called 'fred;' various other machines on its Ethernet could be 'shirl', 'sal', etc. The *fully qualified* name of the 'fred' machine is 'fred.uucp' and must be unique in the `uucp` world; but the full name of 'shirl' is 'shirl.fred.uucp' and its uniqueness is guaranteed by the uniqueness of 'fred.uucp'.

If you only have one machine, your host name and your domain name (within the 'uucp' or 'arpa' top-level domain) are the same.

The above guidelines are valid as of this release date; however, both the Arpa Internet and the uucp network are rapidly evolving. In particular, the Arpa domain is now `.EDU`, `.GOV`, and `.COM` — educational, governmental, and commercial organizations.

Picking a 'Main Machine' for Mail Forwarding

To begin configuration, you must tell `sendmail` whether your system is the 'main machine' in a network, or a 'subsidiary machine' in a network.

If your machine is attached to an Ethernet and is also attached to phone lines, it can be a 'main machine'. Pick one such machine on the network; treat all the rest as 'subsidiary machines' for configuration purposes. Note that if you have a standalone system (a machine which is not attached to an Ethernet), treat it like the 'main machine' in a one-machine network.

If your machine is attached to an Ethernet and you don't have any phone lines, but there is an Arpanet host on your Ethernet, you can pick any workstation as your 'main machine'. If your Arpanet host runs `sendmail`, it can be your 'main machine'. All the other Suns will be subsidiary machines.

Similarly, if you have several machines on an Ethernet, and none of them have phones, pick one as the main machine and leave the rest as subsidiary machines.

A main machine can be aliased as 'mailhost' in `/etc/hosts`.

Configuring for Mail Forwarding

Next, you must define each machine's mailer configuration. The following commands accomplish this. Remember, if your machine is in a server configuration, as per the notes above, the `sendmail.cf` file is `/private/usr/lib/sendmail.cf`.

To configure a 'main machine':

```
# cp /usr/lib/sendmail.main.cf /usr/lib/sendmail.cf
```

To configure a 'subsidiary machine':

```
# cp /usr/lib/sendmail.subsidiary.cf /usr/lib/sendmail.cf
```

Telling Sendmail your Domain Name

To tell `sendmail` what your domain is, edit the file `/usr/lib/sendmail.cf` on the 'main machine' and all the subsidiary machines. Remember that if your machine is in a server configuration, as per the notes above, the `sendmail.cf` file is in `/private/usr/lib/sendmail.cf`. Near the top of the file is a block of lines that looks like this:

```
# local domain names
DDsun
CDsun
```

This defines our domain name as 'sun' within the 'uucp' domain — in other words, 'sun.uucp'. Change these lines to reflect the name you picked. For example,

```
# local domain names
DDfred
CDfred
```

defines your domain name as 'fred.uucp'. The hostname of your main machine (as set up by the `hostname` command) is 'fred', and subsidiary machines, if you have any, will be called (for example) 'shirl.fred.uucp' for a machine whose hostname is 'shirl'.

If your top-level domain is not 'uucp', find the lines that look like:

```
# domain-spec for local domain within universe (e.g., what domains are above?)
DUuucp
```

They should be the next thing in the file. Change `uucp` to your top-level domain name (for example, `DUarpa`).

If you are editing `sendmail.cf` for a subsidiary machine which has phone lines attached to it, you can have `sendmail` route `uucp` mail to certain hosts via the local phone lines, rather than having all `uucp` traffic go through the 'main machine'. To do this, find the lines that look like:

```
# local UUCP connections -- not forwarded to mailhost
CV
```

Put the names of the local `uucp` sites on the end of the `CV` line, or on additional `CV` lines. For example,

```
CV rome prussia georgia
```

This completes the `sendmail.cf` editing for the subsidiary machine. Note that if you have more than one subsidiary with no local `uucp` connections, you can edit the file just once and then copy it to all the subsidiary machines with `rcp`(1). If your server and all clients can share the subsidiary `sendmail.cf`, then just put the file in `/usr/lib` instead of using symbolic links.

Arpanet Forwarding

On your main machine, you can make one optional change. If one of the machines with which you have a `uucp` or Ethernet connection is on the Arpanet and will relay mail for you, look for a block of lines like this:

```
# major relay mailer
DMuucp

# major relay host
DRarpa-gateway
CRarpa-gateway
```

Edit in the mailer that you connect to this host with ('uucp' or 'ether') and the name of the relay host. For example, if you share an Ethernet with a machine called 'cmu-cs-vlsi,' which is on the Arpanet, your entry might look like this:

```
# major relay mailer
DMether

# major relay host
DRcmu-cs-vlsi
CRcmu-cs-vlsi
```

On the other hand, your relay point might be `uucp` host 'ucbvax':

```
# major relay mailer
DMuucp

# major relay host
DRucbvax
CRucbvax
```

This change will let you mail to addresses like 'charlie@mit-ai.arpa' and even though you aren't on the Arpanet, the message will get through. If you don't have an Arpanet relay point, don't worry about this.

Setting up the 'Postmaster' Alias

Edit the file `/usr/lib/aliases` (or `/private/usr/lib/aliases` for a file server or diskless client workstation) and look for the entry for 'postmaster'. This is where people will send mail if they want to get in touch with someone at your site who can deal with mailer problems (you can send mail to 'postmaster's' at other network sites if you have problems with mail that comes from there). The line will look like this:

```
# Following alias is required by the mail protocol, RFC 822
# Set it to the address of a HUMAN who deals with this system's mail problems
Postmaster: root
```

If you manage the mail system for your site, change the name 'root' to the user name you usually log in with, so that messages directed to the Postmaster of your machine will arrive with your usual mail.

If you manage the entire domain, add the alias to the domain wide aliases.

If you manage the mail system for several workstations, change the file on all of them to forward Postmaster mail to wherever you usually read mail. For example, if you are 'henry' on machine 'shirl,' make the line read:

```
Postmaster: henry@shirl
```

You can also create aliases for people or groups of people ('mailing lists') at this time. See the examples in the `aliases` file. You can edit this file now or at any later time. Each time you edit the file, run the `newaliases` program to rebuild the local alias database with your changes. Run `make` in `/usr/etc/yp` on the master `yp` server after updating domain wide aliases.

By default, anytime a message is returned as undeliverable by `sendmail`, a copy of the message header is sent to the Postmaster. If this becomes tiresome, you can disable the feature by editing the `sendmail.cf` file. Find the lines:

```
# CC my postmaster on error replies I generate
OPPostmaster
```

and change them to:

```
# CC my postmaster on error replies I generate
OPnobody
```

Yellow Pages Requirements

Finally, you need to edit the `/etc/hosts` file on your `yp` master server machine so that your main machine ('fred' in this example) will also be known by the alias 'mailhost' to enable mail relay. For example, the `/etc/hosts` file for fred's `yp` domain might look like:

```
192.9.1.1       fred
192.9.1.93      artemis
192.9.1.23      eclipse
192.9.1.2       haley
192.9.1.142     waimea
```

with four subsidiary machines. On the `yp` master server in fred's `yp` domain change the `/etc/hosts` file to look like:

```
192.9.1.1       fred mailhost
192.9.1.93      artemis
192.9.1.23      eclipse
192.9.1.2       haley
192.9.1.142     waimea
```

Store domain wide aliases in `/usr/lib/aliases` on the yp master server.

NOTE *The domain wide aliases software design is fluid as this document goes to press for the Beta release. More information will appear in this section for the First Customer Release.*

Then, while still on the yp master server:

```
# cd /usr/etc/yp
# make
```

This completes the mailer configuration process. (Remember to configure each individual machine!)

Testing Your Mailer Configuration

First reboot all the systems whose configuration files you have changed. Then, send test messages from various machines on the network with a command like this:

```
hal% /usr/lib/sendmail -v </dev/null addresses
```

This will send a null message to the specified address, and display messages about what it is doing. Test that:

- You can send mail to yourself, or other people, on the local machine, by addressing the message to a plain user name ('root', for example).
- If you have an Ethernet, you can send mail to someone on another machine ('root@hobo', for example). Try this in three directions — from the main machine to a subsidiary machine, *vice versa*, and from a subsidiary machine to another subsidiary machine, if you have two.
- If you have a phone line and you have set up a `uucp` connection to another host, you can send mail to someone there and they can send it back (or call you on the phone, if they receive it). Try having them send mail to you. For example, you could send to 'ucbvax!joe' if you have a connection to ucbvax. *Sendmail* won't be able to tell you whether the message really got through — since it just hands it off to `uucp` for delivery — so you have to ask a human at the other end. You might be able to get some idea of what's going on by looking in `/usr/spool/uucp/LOGFILE` (`/private/usr/spool/uucp/LOGFILE` on servers and clients); see the *Uucp Implementation Description* in the *Tutorials* section at the end of this manual.
- Mail something to Postmaster on various machines and make sure that it comes to your usual mailbox, so when other sites send you mail as Postmaster, you're sure you will see it.

Diagnosing Troubles with Mail Delivery

The best tools for diagnosis of mail problems are:

- 'Received' lines in the header of the broken message. These give a trace of which systems the message was relayed through on its way. Note that in the `uucp` network there are many sites that do not update these lines, and that in the Arpanet the lines often get rearranged. You can straighten them out by looking at the date and time in each line. Don't forget to take time zones

into account.

- Messages from 'MAILER-DAEMON.' These typically report delivery problems. More and more, systems produce these messages, rather than throwing away mail they can't deliver.
- The system log, for delivery problems in your group of workstations. Sendmail records what it is doing all the time, and this information is kept in the system log. In the distributed system, the logs are kept for a week, then discarded. Log files are kept in `/usr/spool/log` on your network server machine (the system log configuration is taken care of by the `setup` program during UNIX installation — see `syslog`(8)). Today's log is in file `syslog`; the previous day's is `syslog.0`; two days' back is `syslog.1`, etc. If you have chronic trouble with mail, look at the log once in a while. You might want `cron`(8) to run a shell script nightly which searches the log for SYSERR messages and mails any that it finds to 'Postmaster'. This way, problems are often fixed before anyone notices them, and the mail system runs more smoothly.
- The `mconnect` program can be used to open connections to other sendmail systems over the network.

4.4. Tip

`Tip` establishes a full-duplex connection with another machine, giving the appearance of being logged in directly to the remote computer. You must have an account on the machine you want to connect with.

When you type the `tip` command, it reads commands from the file `.tiprc` in your home directory. This file can be used to alter the default values of `tip` variables. See `tip`(1C) for details and a discussion of the variables used in normal operation. If you use the `-v` flag on the command line `tip` displays the commands as it executes them.

Systems known by `tip`, and their attributes, are stored in the file `/etc/remote`; it describes the remote host(s) that `tip` connects with and stores information that will establish proper connections. See `remote`(5) for details and an explanation of capabilities. `remote` is an ASCII file, structured much like `termcap`(5). Each line in the file provides a description for a single remote system. Fields are separated by a colon (:); lines ending in a backslash character (\) and followed without blank space by a carriage return, are continued on the next line. Each description must end with a colon immediately following the last field.

The first entry is the name of the host system; several aliases may appear in a single name entry, separated by vertical bars (|). The name entry is followed by the fields describing capabilities. Capabilities are of three types: string, numeric, or boolean. String capabilities are given as '*capability=value*', for example: dv=/dev/harris. Numeric capabilities are given as '*capability#value*', for example: br#300. Boolean capabilities are specified simply by listing the capability.

An `/etc/remote` file might look in part like this:

```
decvax|DEC VAX-11/780:\
     :pn=6038842104%:tc=UNIX-1200:
arpavax|ucbarpa|arpa:\
     :pn=6427750%:tc=UNIX-1200:
olympics:\
     :pn=2136815931:tc=UNIX-300:
UNIX-300:\
     :el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:br#300:tc=dialers:
UNIX-1200:\
     :el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:br#1200:tc=dialers:
dialers:\
     :dv=/dev/cua3,/dev/cua2,/dev/cua1:
```

When editing the `/etc/remote` file remember the following rules: 1) every capability must be preceded and followed by a colon; 2) the continuation character (the \ at the end of the line) must not be followed by white space; 3) continuation lines must begin with a tab. These are the same rules you have seen in the `/etc/termcap` and `/etc/printcap` files.

You may need to look at the `remote`(5) man page to determine the meaning of all the capabilities included here.

Characters typed during a `tip` connection are normally transmitted directly to the remote machine. In addition, there is a set of recognized commands that can be given to `tip` on a line that begins with a tilde (~) escape character. These are listed below. Generally, they are for copying files or piping output between the connected machines.

~^D ~.	Drop the connection and exit (you may still be logged in on the remote machine).
~c [*name*]	Change directory to name (no argument implies change to your home directory).
~!	Escape to a shell (exiting the shell will return you to `tip`).
~>	Copy file from local to remote.
~<	Copy file from remote to local.
~p *from* [*to*]	Send a file to a remote UNIX host. When you use the put command, the remote UNIX system runs the command string **cat >***to* while `tip` sends it the *from* file. If the *to* file isn't specified, the *from* file name is used. This command is actually a UNIX specific version of the ~> command.
~t *from* [*to*]	Take a file from a remote UNIX host. As in the put command the *to* file defaults to the *from* file name if it isn't specified. The remote host executes the command string **cat >***from*; **echo ^A** to send the file to `tip`.
~|	Pipe the output from a remote command to a local UNIX process. The command string sent to the local UNIX system is processed by the shell.
~C	Connect a program to the remote machine. The command string sent to the program is processed by the shell. The program inherits file descriptors 0 as remote line input, 1 as remote line output, and 2 as tty standard error.
~#	~# Send a BREAK to the remote system. For systems which don't support the necessary `ioctl` call the break is simulated by a sequence of line speed changes and DEL characters.
~s	Set a variable (see the discussion below).
~^Z	Stop `tip` (only available when run under the C-Shell).
~?	Get a summary of the tilde escapes

Copying files requires some cooperation on the part of the remote host. When a ~> or ~< escape is used to send a file, `tip` prompts for a file name (to be transmitted or received) and a command to be sent to the remote system, in case the file is being transferred from the remote system. For example, typing ~> or ~< will cause the machine to echo `Filename:` and await your input. As in:

```
% ~> Filename:
```

The default end of transmission string for transferring a file from the local system to the remote is specified in the `remote` file, but may be changed by the set command. While `tip` is transferring a file the number of lines transferred will

be continuously displayed on the screen. A file transfer may be aborted with an interrupt. An example of the dialogue used to transfer files is given below (input typed by the user is shown in bold face).

```
arpa% tip monet
[connected]
. . . (assume we are talking to another UNIX system) . . .
ucbmonet login: sam
Password:
monet% cat > foo.c
~> Filename: foo.c
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~< Filename: reply.c
List command for remote host: cat reply.c
65 lines transferred in 2 minutes
monet%
```

Another way to accomplish the same task is shown below:

```
monet% ~p foo.c
. . . (actually echoes as ~[put] foo.c) . . .
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~t reply.c
. . . (actually echoes as ~[take] reply.c) . . .
65 lines transferred in 2 minutes
monet%
```

To print a file locally:

```
monet% ~|Local command: pr -h foo.c | lpr
List command for remote host: cat foo.c
monet% ~^D
[EOT]
. . . (back on the local system) . . .
```

5

Adding Hardware To Your System

Adding Hardware To Your System **137**

5.1. Adding A Board To Your System 138
Kernel Modification 138
File System Modification 139
Trouble Shooting 141
5.2. Connecting Devices To Asynchronous Serial Ports 142
General Theory 142
General Debugging Hints 143
5.3. Adding A Terminal To Your System 145
Serial Port and Cable Connection 145
Kernel Modification 145
File System Modification 146
Problems 148
5.4. Adding A Modem To Your System 149
Serial Ports, Cable Connection And Modem Switches 149
Kernel Modification 150
File System Modification 153
Hayes Specific Considerations 155
Problems 156
5.5. Adding A Printer To Your System 158
Hooking Up A Serial ASCII Printer 158
Hooking Up A Printer To A VPC-2200 Multibus Board 162
Printing On Remote Machines 166

Output Filters 167
Line Printer Commands 168
Access Control 170
Problems 171
Error Messages From The Line Printer System 171

5

Adding Hardware To Your System

This chapter explains many aspects of adding hardware to your system. It covers: how to add a new Multibus board to the system, how to add peripheral devices like terminals and modems to a serial port, and how to add a printer to the system.

In most cases adding new hardware to your system is a three tiered process: you connect the actual hardware piece as appropriate, if necessary you reconfigure the UNIX† kernel, and you edit files on your system to adapt it to the new device. For each kind of new hardware device, the process is explained in the appropriate section below.

The first kind of hardware device covered is a circuit board. You will probably have to reconfigure your kernel when a board is added.

Next, a section explains general procedures for adding devices to asynchronous serial ports, followed by specific instructions for terminals and modems.

Finally, we explain how to add a printer, and give some explanation of the line printer spooling system as implemented under Sun UNIX. Printers may be hooked-up on a serial port or connected directly to a controller board that drives a given model; each case is covered.

† UNIX is a trademark of AT&T Bell Laboratories.

5.1. Adding A Board To Your System

When a new board is added to any Sun system, at least three things must be done: the board itself must be inserted in the computer's card cage; a program called a device driver must be added to the UNIX kernel; and the UNIX file system must be modified to accept the new device.

This section will explain the last two procedures, adding a device driver and modifying the file system. For each board Sun supports, there is an explanation of card cage installation in the *Hardware Installation Manual* for your Sun system.

Kernel Modification

You must be superuser to make the following kernel modifications.

The UNIX kernel contains programs called device drivers that help the operating system pass information between the system hardware and the system software. UNIX allows users to specify which devices they want to include in the kernel through a process called configuring the kernel. The more device drivers a user includes in the kernel, the more space it occupies in main memory, so it is important to run a streamlined kernel that includes only those device drivers you need for your system. Adding a new board requires the addition of a device driver for it in the kernel.

To specify which devices should be included, the user edits a configuration file. Most lines in the configuration file represent device drivers. We are concerned here only with device driver lines for the boards you might install in your system.

Below we give an example of an entry in the kernel configuration file for a Sun-1 color board. To add a color board device driver to the kernel, the line is edited into the kernel configuration file, and the system is reconfigured.

```
device  cgone0 at mb0 csr 0xe8000 priority 3
```

Each board in the system requires an entry of this type in the kernel configuration file. If you look at the generic kernel configuration file distributed with the system, `/sys/conf/GENERIC`, (or a printed copy of it in the *Installing UNIX Manual*), you will find a kernel configuration entry for each device Sun currently supports. For further explanation, each device listed in `/sys/conf/GENERIC` is documented in *Section 4, Special Files*, in the *System Interface Manual*. For example, if we looked up `cgone` in the manual, we would find out, among other things, what values are mandatory for specified device driver fields, and how much Multibus memory the board consumes.

For boards that Sun does not support, you will need to write your own device driver and place an entry for it in the configuration file. This procedure is described in *Writing Device Drivers For The Sun Workstation* in the *System Internals Manual*. It requires expert understanding of the kernel.

When you have determined device driver information that must be added to the kernel configuration file for the new board, you will need to do the steps shown below to complete the kernel reconfiguration process. These steps explain the process on a system named GRENDEL:

1) Change to `/sys/conf` directory and make a copy of the current kernel configuration file to keep until the new one is installed and running.

```
# cd /sys/conf
# cp GRENDEL old.GRENDEL
```

2) Edit the kernel configuration file, adding a device driver entry for the board you have installed into the card cage. Look in *Section 4* of the *System Interface Manual* for specifications for the boards that Sun supports.

3) Run `/etc/config` on the new kernel configuration file.

```
# /etc/config GRENDEL
```

4) Build the new kernel.

```
# cd ../GRENDEL
# make depend
# make
```

5) Install the new kernel and try it out.

```
# cp vmunix /newvmunix
# /etc/halt
> b newvmunix -s
```

6) If the new kernel appears to work, save the old kernel and install the new one in `/vmunix`.

```
# cd /
# mv vmunix ovmunix
# mv newvmunix vmunix
# /etc/reboot
```

Remember to remove the `old.GRENDEL` kernel configuration file you created as a backup.

For more information about kernel building, see the manual *Installing UNIX On The Sun Workstation.*

File System Modification

You must be superuser to make the following system modifications.

The UNIX kernel communicates with devices through a special file in the directory `/dev`. When a new board is added to the system, a new entry for it must be made in the `/dev` directory. This is relatively easy to do with a shell script called `MAKEDEV` (8), located in `/dev`. `MAKEDEV` takes an argument that is the device-name of a device supported by the system, and automatically installs the files for that device in the `/dev` directory.

For example, if you are adding a Sun-1 color board, you will need to run `MAKEDEV` like this:

```
# cd /dev
# MAKEDEV cgone
```

The following table gives a list of the arguments MAKEDEV accepts for the boards supported by Sun. It also includes the name of the device driver that is entered into the kernel configuration file — the MAKEDEV argument and the device driver name are often identical, but in several cases are different so be attentive. Where an asterisk appears, it means that a logical number — '0' being the first, '1' being the second, and so forth — should be appended.

Table 5-1 *Sun Supported Boards*

MAKEDEV Argument	GENERIC File Device Driver	Description of Device
Tape Controller Boards:		
tm*	mt*	Tapemaster 1/2" tape
ar*	ar*	Archive 1/4" tape
st*	st*	SCSI (Archive) 1/4" tape
xt*	xt*	Xylogics 472 1/2" tape
Disk Controller Boards:		
ip*	ip*	Interphase 2180
xy*	xy*	Xylogics 450
sd*	sd*	SCSI Disk
Terminal Multiplexor Boards:		
ttys	zs2-3	First Multibus SCSI board UARTS ttys0-ttys3
ttyt	zs4-5	Second Multibus SCSI board UARTS ttyt0-ttyt3
mti*	mti*	Systech MTI-800A/1600A
Ethernet Controller Boards:		
ec*	ec*	3COM Ethernet Board
ie*	ie*	Sun-2 Ethernet Board
Printer Boards:		
vpc*	vpc0	Versatec & Centronics (Systech VPC-2200)
vp*	vp0	Versatec (Ikon interface)
Graphics/Windows Boards:		
cgone*	cgone*	Sun-1 color graphics board
cgtwo*	cgtwo*	Sun-2 color graphics board
bwone*	bwone*	Sun-1 black & white graphics board
bwtwo*	bwtwo*	Sun-2 black & white graphics board
Miscellaneous Boards:		
sky	sky0	Sky FPP board

After MAKEDEV, you can do an `ls -l` to check the status of the new device. MAKEDEV takes care of setting all permissions and ownerships properly. You can read through the shell commands in MAKEDEV if you have questions about

what it does.

Trouble Shooting

Here is a general list of hints for debugging new boards.

1) Check the hardware. Re-seat the board to assure electrical contact. Check hardware manuals to make sure all switch settings are proper.

2) If you boot the system with a copy of the GENERIC kernel after installing a device that Sun supports, and you still have the problem, chances are good that it is a hardware problem. If the problem disappears using GENERIC, you should begin looking in the kernel or the file system for the problem.

3) Make sure the new device is included correctly in the kernel configuration file. One way to ascertain this is to look in a file called `/usr/adm/messages`, which collects all the messages generated during boot-up. By looking at the last boot-up listing in `/usr/adm/messages`, you can determine positively whether the device was included in the kernel last used to boot the system. It should appear in the boot-up listing as one of the devices the boot-up procedure found. You may also use `dmesg` (8) by hand to see recent diagnostic messages — `/etc/dmesg` is run to produce the file `/usr/adm/messages`.

4) Do an `ls -l` in the `/dev` directory to make sure the new device is installed. The `ls -l` will also show the permissions and major and minor numbers for each device present. Check the permissions for the device against those found in the section of the `MAKEDEV` script that installs the device in question, they should match. Also make sure the device major and minor numbers match those in `MAKEDEV`.

5) Try the simplest commands to see if ANYTHING is working. The closer you come to isolating the spot where the system breaks down, the fewer places you need to look to make fixes. For example, see if a new printer board works by simply 'catting' a file to it:

```
% cat myfile > /dev/lp
```

rather than trying to pipe a job through text processors to `lpr`, where the job could fail for a reason that has nothing to do with the new board.

5.2. Connecting Devices To Asynchronous Serial Ports

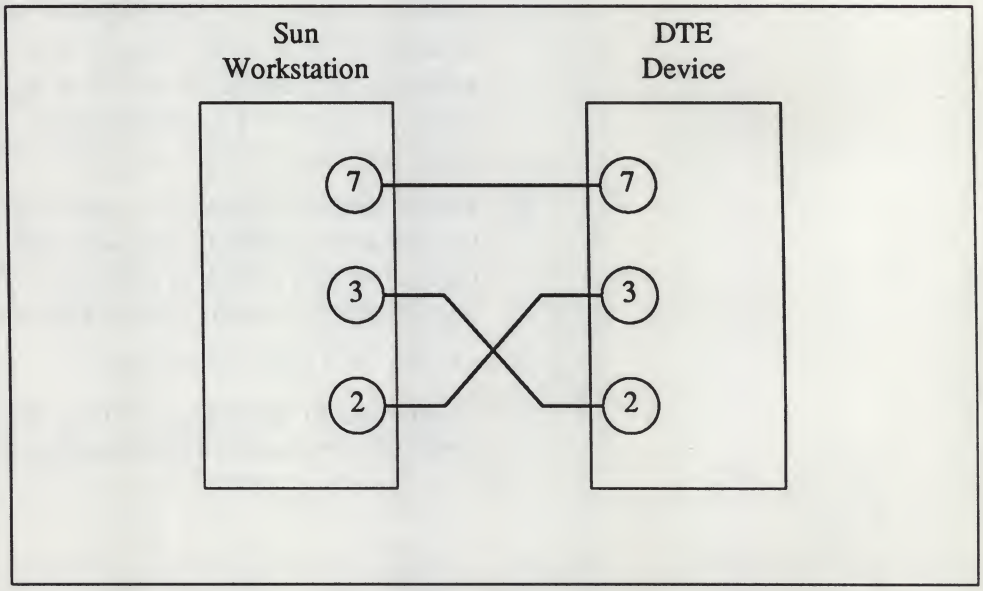
Please read the section called *Asynchronous Serial Ports* in the *Hardware Installation Manual* for your machine before proceeding further. There you will find a discussion of serial port signals for your machine. Become familiar with the hardware specifications for your Sun model.

General Theory

This section introduces the general topic of connecting equipment to serial ports and discusses some of the terminology. For specific treatment of how to add terminals, modems or printers see those sections below.

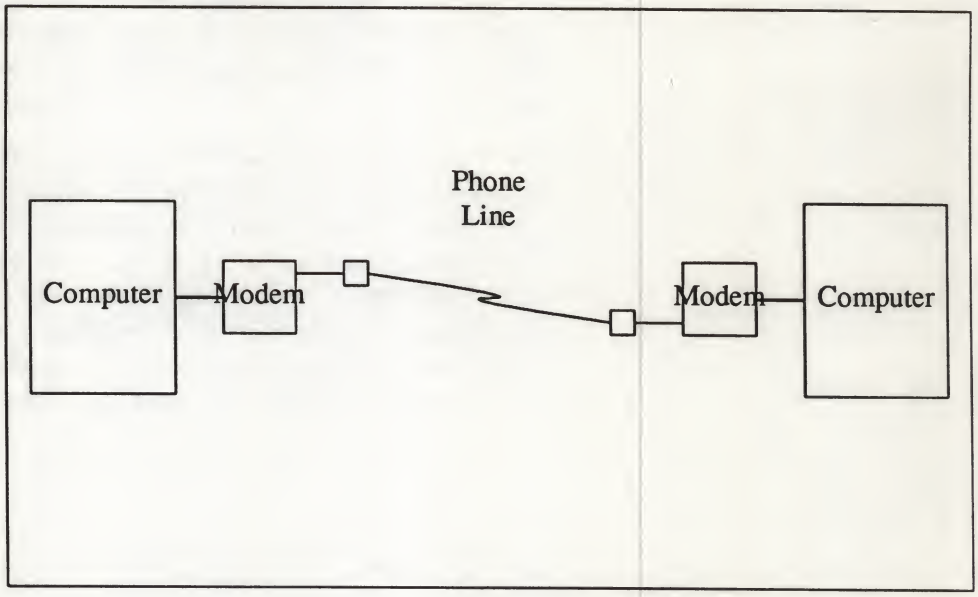
Put simply, a serial port sends a byte of information bit by bit over a single line. All of the Sun serial ports have RS-232-C cabling conventions, but use RS-423 signaling. You may attach modems, terminals, printers, plotters, or other peripheral serial devices which accept the RS-423 signaling, to the serial port connectors on your machine. All ports on Sun machines are wired as DTE — data terminal equipment. This means that the data transmit signal from the port is on pin 2 and the receive data from the peripheral is on pin 3. To connect a Sun Workstation directly to a terminal, printer, computer, or other DTE device, use a null modem cable that crosses lines 2 and 3, thereby enabling the proper signal connection at the other end. See Figure 1-1 below.

Figure 5-1 *Null Modem*



The Sun Workstation is connected to phone lines by the addition of a modem. The modem can be connected directly to the serial port. Modems are wired as DCE (data communication equipment) devices. A DCE receives serial data from the DTE on pin 2 and sends it down the telephone line; a DCE receives telephone data and sends it out on pin 3 to the DTE. See Figure 1-2 below for a depiction of how two computers would transmit signals through modems along cables and phone lines.

Figure 5-2 *Communications Through Modems*



When you connect a peripheral device to a serial port on the Workstation, make sure the cable connection between the serial port and the device is properly installed. Next make sure the appropriate serial device is configured in the UNIX kernel. Note: the generic kernel contains all the Sun supported device drivers. If you have re-configured or customized the kernel, you may need to add back the appropriate serial driver. See the section on *Kernel Configuration* in the manual *Installing UNIX On The Sun Workstation.* If specific kernel changes are necessary, they are discussed in the following sections.

You always need a special file in `/dev` for the UNIX kernel to run any device. There is an easy way to make this special file: you use the `/dev/MAKEDEV` command, which installs a special file for new UNIX devices. The command is explained in `MAKEDEV` (8).

General Debugging Hints

Here we discuss two general areas where problems occur when you have added new peripheral devices to the system.

Hardware Changes — If you have added new hardware and there are problems, check the cabling and connections first. Is everything tight? Once again verify that the new device accepts the RS-423 communication protocol, and consult the product manufacturer's manual to make sure it has been installed properly. Are the workstation and the new device set up for compatible communication? The Sun defaults to the following: 7 data bits, even parity, 9600 baud rate, flow control enabled (xon/xoff). Check which control signals the other equipment expects.

Next, check the condition of the cable. Here you may want to refer once again to your Sun Workstation *Hardware Manual* to determine what signals are sent from and received at specific ports on your board. To check for cable problems it is helpful to have a "breakout box". A breakout box plugs into the RS-232 cable, providing a patch panel that allows you to connect any pin to any other pin(s); it will often contain LEDs which display the presence or absence of a signal on

each pin. Sometimes a cable has become corrupt and signals are being flipped randomly. In this case a line may receive when it should transmit, or the reverse. These suggestions should give you a start on troubleshooting. Problems specific to terminals are covered in the *Adding Terminals* section below.

New Software — If you have problems after installing a new system kernel along with the new peripheral device, there could be something wrong with the new kernel. This is likely to be a total non-response symptom rather than the intermittent weirdness of a bad line. Back out the new kernel and install the GENERIC kernel in its place, then boot the system with GENERIC. If the problem disappears with GENERIC, you have a problem with the new kernel — possibly you did not add the device driver, or did not add it properly. If the problem persists with GENERIC, it is probably in the cabling or the hardware.

5.3. Adding A Terminal To Your System

Please read the section above called *Connecting Devices To Asynchronous Serial Ports* before proceeding further. The general discussion there will prepare you to install a new terminal on your system.

This section explains the steps necessary for adding a terminal, and gives some hints on what to do if you run into problems.

Serial Port and Cable Connection

We begin with the hardware implementation. Make sure you have an available serial port on your workstation. The number of serial ports varies from one Sun model to another. Each Sun workstation provides at least two asynchronous serial ports controlled by the Serial Communications Controller on the Sun-2 CPU board. If all the existing serial ports are in use, and you are not planning to disconnect any current peripheral devices to make a port available, you will need to add a new Multibus board to the card cage to increase the number of serial ports. The instructions for adding a Multibus board are given above in the section *Adding A Board To Your System.*

When you do have a port available, connect the terminal to the workstation with a null modem cable. A basic null modem cable swaps lines 2 and 3 so that the proper transmit and receive signals are communicated between two Data Terminal Equipment (DTE) devices. Line 7 goes straight through.† Make sure all the connections are secure and check control signals.

Kernel Modification

You must be superuser to make kernel modifications. If you are adding a first terminal, make sure that the system kernel contains a `zs0` device driver to run the CPU ports (this is standard); if you are adding terminals to a SCSI board make sure the proper `zs*` driver entry is in the kernel configuration file; if you are adding terminals to a Systech multi-terminal interface board (MTI), make sure the proper `mti0` driver entry is in the kernel configuration file.

If you had to add a new board to make ports available for your terminals, you have probably made necessary kernel changes already, as outlined above in the section *Adding A Board To Your System.* That section explains how to edit and reconfigure the system kernel.

Use `dmesg` (8) to look at system diagnostic messages to make sure the system is configured they way you want it.

Look in your kernel configuration file in the `/sys/conf` directory to find the device drivers for which your system is configured. An explanation of the device driver entries in this file is given on the appropriate page in *Section 4* of the *System Interface Manual.* For terminal connection, make sure the `flags` bit is set for a hardwired line on the port — set to on.

† The Systech MTI board requires pin 5 asserted. The simplest way to get this is to connect pins 4 and 5 at the connector going to the MTI port.

File System Modification

UNIX needs a special file in /dev to run any device. For serial ports, these are known as /dev/tty* — where the asterisk is a letter or number value or both. When you add a board you have to run the MAKEDEV shellscript to install the special files for it in /dev. If you are connecting to available ports on a particular board already in use, then the MAKEDEV has already been run and you can ignore the following information.

The MAKEDEV command takes an argument which is the device name for the device in the kernel. In the case of a serial port board, one /dev/tty* file is created for each terminal interface on the board. The table shows examples of the MAKEDEV command for the Multibus boards that Sun supports, and the names of the entries MAKEDEV creates in /dev.

Table 5-2 *Using* MAKEDEV *For New Boards*

Type	*Command*	*Devices*
Sun CPU Board	MAKEDEV std	ttya,ttyb
1st Multibus SCSI Board	MAKEDEV ttys	ttys0 — ttys3
2nd Multibus SCSI Board	MAKEDEV ttyt	ttyt0 — ttyt3
Systech MTI-800 Board	MAKEDEV mti0	tty00 — tty0f
Systech MTI-1600 Board	MAKEDEV mti0	tty00 — tty0f

Next you will need to edit two files that tell the system about the new terminal. First edit /etc/ttys, a file which contains terminal initialization data. The /etc/ttys file is read by the init program and specifies which serial ports will have a login process created for them.

There should be one line in /etc/ttys for every serial port. The name of the port in /etc/ttys is the same as its name in /dev. The /etc/ttys file looks in part like this:

```
12console
12ttya
02ttyb
12ttys0
14ttys1
```

The first character of a line in the /etc/ttys file is either a '0' or a '1'. If '0', the init program ignores that line and no logins can occur on that line; if '1', the init program creates a login process for that line.

The second character on the line is used as an argument to getty, which performs such tasks as setting the baud-rate, reading the login name, and calling login. You only need to be concerned with the baud-rate, since different values here signify nothing more than different baud-rates for the terminal initialization. In the file /etc/gettytab you will find the values your system recognizes for different baud-rates. For example,

```
2|std.9600|9600-baud:\
    :sp#9600
3|std.300|300-baud:\
    :sp#300
4|std.1200|1200-baud:\
    :sp#1200
```

is a `gettytab` entry for initialization of terminals at three different baud-rates. Therefore, a '2' as the second character on the line would set the rate to 9600, an 'f' to 1200. If we were putting a terminal on /dev/ttys2 at baud-rate 9600, we would add the following line to the example `/etc/ttys` file shown above:

```
12ttys2
```

at the end of the file. This terminal will have a login process started for it at 9600 baud. After editing the `/etc/ttys` file you **must** notify `init` which then reads the new information in `/etc/ttys`. The command is:

```
# kill -1 1
```

Now edit the file, `/etc/ttytype`. This file is a database containing information about the kind of terminal attached to each tty port on the system. `/etc/ttytype` has one line per port, containing the terminal type, a blank space, and the name of the tty. For example,

```
sun console
925 ttya
925 ttyb
925 ttys0
925 ttys1
dialup ttyd0
dialup ttyd1
dialup ttyd2
dialup ttyd3
```

indicates that the terminal attached to ttys1 in the example above is treated by the system as a TVI 925, as are all other terminals attached to this system. The console is treated as a Sun Workstation. The type 'dialup' is for phone-in lines available in this example. To find the kind of terminal types your system recognizes, look in the file `/etc/termcap` (and see `termcap`(5)); this is a database describing the capabilities of terminals and the way operations are performed on them. If your terminal type is found in `termcap`, use it. If your new terminal has an emulation mode for one of the terminal types found in your `termcap` file, edit that name into `ttytype` and set the corresponding emulation mode on the terminal. If your terminal type is not in `termcap`, you may create your own `termcap` entry. The information in `ttytype` is read at login time by `tset` and `login` to initialize the TERM variable.

Finally, you will need to set some switches on the terminal so it can communicate with the Sun. Check to see if you have changed any settings on the Sun from the defaults given here: 7 data bits, even parity, flow control enabled (xon/xoff); set baud rate to the same rate you set in the `/etc/ttys` file.

Problems

If you have problems after installation, check the cabling first, as outlined above in the section *Connecting Devices To Asynchronous Serial Ports*. Next, check the information in the `/etc/ttys` and the `/etc/ttytype` files, and make sure you have restarted `init` as explained above.

If the terminal behaves strangely, check the `/etc/termcap` file. You may have unexpected whitespace, or you may have forgotten to escape the newline character in your entry: a backslash (\) followed immediately by a newline (carriage return) will continue a line onto the next.

If you have not set the tty type in the `ttytype` file, you can set it at your terminal with the command:

```
# setenv TERM sometype
```

There are several ways to check aspects of your terminal environment. The `stty` command typed with no argument will tell you the baud-rate of the terminal and the settings of options which are different from their defaults. The `all` argument reports all normally used option settings. The `everything` argument reports everything that `stty` knows about.

The `printenv` command with no arguments prints out the variables in the environment.

You can type `set` with no arguments, and it will report the value of all shell variables.

5.4. Adding A Modem To Your System

Please read the section above called *Connecting Devices To Asynchronous Serial Ports* before proceeding further. The general discussion there will prepare you to install a modem on your system.

This section explains the steps necessary for adding a modem, and gives some hints on what to do if you run into problems.

Sun supports three currently manufactured modems: the Ven-Tel 1200-plus, the Ven-Tel 1200-32, and the Hayes Smartmodem 1200. Ven-Tel modems are run in Hayes compatibility mode. Sun will continue to support the discontinued Ven-Tel MD 212 series, although it alone is not discussed below.

Where this section refers to systems that have both auto-dial and auto-answer capabilities, the references are only to systems after Sun release 1.2.

Serial Ports, Cable Connection And Modem Switches

Serial Ports — Make sure you have an available serial port on your workstation. The number of serial ports varies from one Sun model to another. Each Sun workstation provides two asynchronous serial ports controlled by the Serial Communications Controller on the Sun-2 CPU board. If you also have a Sun-2 SCSI board in your system, two additional SCC's make four additional serial ports available. If all the existing serial ports are in use, and you are not planning to disconnect any current peripheral devices to make a port available, you will need to add a new Multibus board to the card cage to make more ports available. The instructions for adding a Multibus board are given above in the section *Adding A Board To Your System.*

Cable Connection — If you do have a port available, connect the modem to the workstation with an RS-232 cable that has pins 2 through 8 and pin 20 wired straight through. You may also use a 25 pin ribbon cable to connect the ribbon to the system. Make sure all the connections are secure.

The Systech MTI board requires a special cable with pins 2, 3, 4, 5, 7, and 20 wired straight through, and pin 6 on the MTI end wired to pin 8 at the modem end.

Modem Switches — There are differences between the Ven-Tel modems and the Hayes Smartmodem. However, with either brand the switch settings shown below work for use with `tip`(1) and `uucp`(1). Always read the manufacturer's manual before attempting to adjust equipment.

First the Ven-Tel 1200-plus models. (Note that this information does not apply to the old Ven-Tel MD 212.) There is one panel of external switches and one of internal switches on each of the Ven-Tel models discussed here. For the external panel, set all switches to the open or up position. When you open the modem box you will see another panel of switches, which should be set as follows:

- Switch 1 up for hardware DTR.
- Switch 2 up for WECO control signals.
- Switch 3 down to get Hayes protocol.
- Switch 4 can be set according to your preference. Set it up to disable the speaker so you will not hear the high-pitched carrier noise when connecting;

set it down to enable the speaker.

The Ven-Tel 1200-32 is somewhat different. Here are the settings for its internal 10-section dip switch:

- Switch 1 off for hardware DTR.
- Switch 2 on for numeric result codes.
- Switch 3 on to send result codes.
- Switch 4 on to not echo commands.
- Switch 5 off to answer incoming calls.
- Switch 6 off for hardware carrier detect.
- Switch 7 off — not used.
- Switch 8 off — not used.
- Switch 9 off for no hardware DSR.
- Switch 10 controls the speaker as with the Ven-Tel 1200-plus.

Next the Hayes Smartmodem. The proper switch settings are given below. There is only one switch panel on the Hayes; down is `on` and up is `off`.

- Switch 1 up for hardware DTR.
- Switch 2 down for numeric result codes.
- Switch 3 down to send result codes.
- Switch 4 down to not echo commands.
- Switch 5 up to answer incoming calls.
- Switch 6 up for hardware Carrier Detect.
- Switch 7 up for connection to RJ11 modular jack.
- Switch 8 down enable command recognition.

After changing the switch settings on any model, you must power cycle the modem by unplugging it from the electrical outlet, waiting a few seconds, and then plugging it back in. Looking at the front of a properly installed modem, you should see lights AA and TR lit up when the modem is not in use; and the lights should be lit or blinking when the modem is in use.

Kernel Modification

You must be superuser to make the following kernel modifications.

The UNIX kernel contains programs called device drivers that allow the operating system to pass information between system hardware and system software. Users may specify which devices they want to include in the kernel through a process called configuring the kernel. To specify which devices should be included in the system, and exactly how those devices should operate, the user edits a configuration file.

You must edit the device driver specification for the serial communications controller in your configuration file, to enable carrier detect on the line where the modem is attached. We refer to this as turning `off` the software carrier, which then allows the system to detect a hardware carrier on the line. When turned `on` the system treats each line as hard-wired with carrier always present. Turning `on` or `off` is done by specifying a parameter in the `flags` field. For example, below is a sample entry from a kernel configuration file for the serial communications controller on the standard Sun CPU board:

```
device  zs0 at mb0 csr 0xeec800 flags 0x3 priority 2
```

To make the device in this example into a modem compatible driver, the hexadecimal value '0x3' after `flags` will have to be changed. The same field will have to be changed for other device drivers if your modem is going to be attached to a board other than the Sun CPU board.

Specifically, you need to make the following changes to the device driver specification line in the kernel configuration file. The changes for each of the three boards to which a modem can be attached are given below.

The Standard Sun CPU Board — The serial communications controller on the standard Sun CPU board, device `zs0`, provides two lines with full modem control. For both dial-in and dial-out on the same serial port, the `flags` bit in the kernel corresponding to that serial port has to be set to zero. This enables hardware carrier detect so that the Sun can tell when someone dials in or hangs up. The default value of `flags` for `zs0` in the Generic kernel is 3, indicating software carrier for both ports a and b. To permit hardware carrier detect on `ttya`, `flags` should be changed to '0x2', on ttyb to '0x1', and on both to '0x0'.

A Multibus SCSI Board — Each Multibus SCSI board has two serial communications controllers identical to the one on the CPU board, for a total of four lines with modem control on each SCSI board. These are devices `zs[2,3]` for the first board and `zs[4,5]` for the second. Set the `flags` value for each driver according to the rules explained above for the CPU board. Remember that `zs2` is for the lines `ttys[0,1]` and that `zs3` is for the lines `ttys[2,3]`; if you have a second SCSI board the ttys are, by convention, `ttyt[0-3]`.

A Systech MTI Board — The Systech MTI-800 and MTI-1600 boards require a kernel configuration line for device `mti0`. As above, the `flags` value is set in hex. The default `flags` value on installation of either MTI board is '0xffff'. This value selects software carrier detect for all lines. For lines with modems, you need hardware carrier detect and must set the corresponding `flags` bit in the kernel to zero, just as in the examples above.

The following example shows how to determine the proper hexadecimal values for the `flag` bits on a Multibus board, it visualizes a Systech MTI-1600 board where ports zero through 7 are turned on, software carrier detect, and ports 8 through f are turned off, hardware carrier detect for modem operation. The logic here can be applied to all Multibus boards.

port	f	e	d	c	b	a	9	8	7	6	5	4	3	2	1	0
on									1	1	1	1	1	1	1	1
off	0	0	0	0	0	0	0	0								

If we take the values shown here, '0000' '0000' '1111' '1111', and convert to hexadecimal, we arrive at the entry for the `flags` parameter: '0x00ff'.

For more information on the serial communications driver and the multi-terminal interface, see the pages `zs`(4s) and `mti`(4s) in the *System Interface Manual.*

Here are the steps you follow to edit and install the new kernel on a system named `CHAOS`:

1) Change to `/sys/conf` directory and make a copy of the current kernel configuration file to keep until the new one is installed and running.

```
# cd /sys/conf
# cp CHAOS old.CHAOS
```

2) Edit the kernel configuration file, changing the `flags` bit, as explained above, for the board to which you will attach the modem. For example, if you want to put the modem on `ttya`, change an entry like this:

```
device  zs0 at mb0 csr 0xeec800 flags 0x3 priority 2
```

to an entry that looks like this:

```
device  zs0 at mb0 csr 0xeec800 flags 0x2 priority 2
```

3) Run `/etc/config` on the new kernel configuration file.

```
# /etc/config CHAOS
```

4) Build the new kernel.

```
# cd ../CHAOS
# make depend
# make
```

5) Install the new kernel and try it out.

```
# cp vmunix /newvmunix
# /etc/halt
> b newvmunix -s
```

6) If the new kernel appears to work, save the old kernel and install the new one in `/vmunix`.

```
# cd /
# mv vmunix ovmunix
# mv newvmunix vmunix
# /etc/reboot
```

Remember to remove the `old.CHAOS` kernel configuration file you created as a backup.

For more information about kernel building, see the manual *Installing UNIX On The Sun Workstation.*

File System Modification

You must be superuser to make the following system modifications.

Using mknod — First you create an alternate device for your modem in the `/dev` directory by using `mknod`. This alternate device allows a single tty line to be connected to a modem and used for both incoming and outgoing calls.

There are several arguments which you must give to `mknod`. The first is the name of the device you are making. It will be `cua*`, where the asterisk is the logical value of this alternate device on your system — 0 for the first alternate device, 1 for the second and so forth. The second argument will always be `c`. The last two arguments are the major and minor device numbers. To determine what numbers to use here do an `ls -l` on the `/dev/tty*` device you are making the alternate device for. If you are going to use the first port on your CPU board — `ttya` — do the following:

```
# ls -l /dev/ttya
crw--w--w-  1 root       12,   0 Sep 17 18:27 /dev/ttya
```

Here the major device number is '12' and the minor device number is '0'. When running `mknod` to create an alternate device for a modem you always give the actual major device number and you always add 128 to the minor device number. Thus, in this example, the `mknod` command would take as its last two arguments '12 128'. You also change the mode and ownership of this device. The whole process would look like this:

```
# cd /dev
# mknod cua0 c 12 128
# chmod 600 cua0
# chown uucp cua0
```

In this example we created alternate device `cua0`, the special file for the first modem attached to a system. The second modem attached would be `cua1`, and so forth.

While still in the `/dev` directory, move the tty device whose software carrier you turned off in the `flags` field of your device driver specification in the kernel above, to a dialing device. This is the same one for which you created the alternate device with `mknod`. In this example we move the first port on the CPU board to the first modem device.

```
# mv ttya ttyd0
```

The second modem device would be `ttyd1`, and so forth.

Updating ttys — Now edit the `/etc/ttys` file, adding an entry for the new `ttyd1` line, and disabling logins for the no longer existing `ttya` device.

A modem suitable for use at both 1200 and 300 baud could use a `gettytab` entry similar to this:

```
# fast dialup terminals, 1200/300 rotary (can start either way)
#
3|D1200|Fast-Dial-1200: :nx=D300:fd@:tc=1200-baud:
5|D300|Fast-Dial-300:   :nx=D1200:tc=300-baud:
```

The resulting /etc/ttys file might look like this:

```
12console
02ttya
02ttyb
13ttyd0
```

The ttya line has logins disabled. The ttyd0 line has logins enabled at either 1200 or 300 baud — sending a break through the modem causes getty to switch to the other baud rate. If you want to use a modem at a single baud rate only, set up /etc/ttys as you would a directly connected terminal. For example, this gettytab entry

```
f|std.1200|1200-baud:   :fd#1:sp#1200:
```

can be used for a fixed 1200-baud line. Modify /etc/ttys accordingly:

```
12console
02ttya
02ttyb
1fttyd0
```

After you have finished editing /etc/ttys, you must re-initialize the file by notifying init with the kill command:

```
# kill -1 1
```

The /etc/ttytype file should also be updated. The previous section *Adding A Terminal To Your System* discusses this process and the related files in more detail.

The /etc/remote File — The file /etc/remote stores the attributes of systems that you dial out to using tip (1). See the *Tip* section of the *Communications* chapter of this manual. It is structured somewhat like termcap. You must now edit that file. Each line in the file provides a description for a single known system. Fields are separated by a colon (:). Lines ending in a backslash (\) and followed immediately by a newline (carriage return) are continued on the next line. Look in the /etc/remote file or see the remote (5) manual page in the *System Interface Manual*, for further explanation. The example given below would work to connect to a system named 'sunphone', at phone number 7630927, at baud-rate 1200, using the 'hayes' auto call unit protocol, on dialer 'cua0':

```
sunphone:\
     :pn=7630927%:tc=UNIX-1200:
UNIX-1200:\
     :el=^D^U^C^S^Q^O@:du:at=hayes:ie=#$%:oe=^D:br#1200:tc=dialers:
dialers:\
     :dv=/dev/cua0:
```

Install `uucp` — Once your modem is installed and working, you may install `uucp`. See the appropriate sections in the *Communications* chapter of this manual.

Set Up The Mail System — Finally you must check to make sure that your machine's mailer configuration is set up properly. See *Setting Up The Mail System* in the *Communications* chapter of this manual for a complete discussion of the mail system. If you are adding a modem to connect a main machine to phone lines, or if you are a standalone machine not on a local network, install the `main` file in `sendmail.cf`. If you are a subsidiary machine on a local network (standalone or client) and planning to stay that way, despite the attachment of phone lines, install the `subsidiary` file in `sendmail.cf`.

Hayes Specific Considerations

`tip` **Support** — To use a Hayes modem with `tip`, you should specify the modem type in the `/etc/remote` file as either `at=hayes` or `at=at`. The phone number attribute (`pn`) can contain any valid dial commands; see your modem manual for details. The most common commands to the phone number attribute are:

- A phone number of numeric characters 0-9.
- A comma (,) will cause a 2 second pause to wait for a secondary dial tone. For example to dial an outside line from a local phone network.
- A `P` or `T` will switch to pulse or tone dialing. By default tip will use tone dialing. Typically, a rotary phone has pulse dialing and a push button has tone.
- You may set parameters for the dialer by starting the phone number with an 'S' (for set) flag. Read the Hayes manual for an explanation.
- Note that `tip` can cycle through a set of phone numbers, dialing each one until a connection is made. Previously the numbers were often separated with a comma, although this feature was never documented. Now, since comma is a valid dial character, phone numbers must be separated with a vertical bar (|), just like the separator in the name field. For example, the `/etc/remote` line for `pn` would look something like:

  ```
  :pn=2138896565|2138896564|2138896563:
  ```

`uucp` **Support** — To use a Hayes (or AT) modem with `uucp` the modem type in the `/usr/lib/uucp/L.sys` file should be `ACUHAYES`.

- The `uucp` default is for tone dialing.
- The phone number may contain any valid dial commands, however `uucp` uses any alphabetic prefix of a phone number to look up a translation in the

`L-devices` file. Therefore, you must insert a minus (-) before a phone number that begins with a letter. An `L.sys` line that uses a Hayes modem to call with pulse dialing might look like:

```
adiron Any ACUHAYES 300 -P7620883 login:-EOT-login: uucp ssword: junk
```

To use the default of tone dialing, simply omit the `-P`.

- As in `tip`, you may set parameters for the dialer by starting the phone number with an 'S' (for set) flag. Read the Hayes manual for an explanation.

Problems

- If you have problems after installation, check the cabling first: as outlined above in the section *Connecting Devices To Asynchronous Serial Ports*, and make sure you have pins 2 through 8 and 20 wired straight through. (If MTI, wire 2, 3, 4, 5, 7, and 20 straight through and have pin 8 on the modem go to pin 6 on the MTI port.) Next, check the information in the `/etc/ttys` and the `/etc/remote` files, and make sure you have restarted `init` as explained above.
- If you cannot access a port, and find a process running on it when you do a `ps -ax`, then make sure you have the 8 pin connected in your cable. If that does not work, check to make sure your device driver is configured properly to set the correct flag for the line to `off`.
- Sometimes even when both the hardware and software are correct, the device driver will get into a state where it will not let the alternate port be opened. You must do a `kill -1 1` to notify `init` and reset the flags on the device driver.
- If you get a `can't synchronize with hayes` error message when using a Hayes-compatible modem, check internal and external modem switch settings, and check the the cable connection. Power cycle the modem if necessary.
- If you get a `can't synchronize with ventel` error message when using a Hayes-compatible modem, look in the `/etc/remote` file and make sure you have changed `at=ventel` to `at=hayes`.
- The message `tip: /dev/cua0: No such device or address` usually means that the device special file is missing from `/dev` (or is incorrect there), or the device driver is missing from your kernel.
- The message `all ports busy` may mean that the port is actually busy running `tip` or a dial-in user. You can do a `ps ax` to see what is running. You should not see a `getty` on the port, except momentarily as the port receives a call. If you do see a `getty` with no user dialed in, you have not set up carrier detect properly. Check: the flags bit in the kernel; the cable; the modem settings; the `etc/ttys` file for correct device entry. If no process is currently using the serial port, there may be a leftover lock file. Look for a lock file in `/usr/spool` and`/usr/spool/uucp` and remove it. It will look something like `LCK.cua0`. Make sure that internal switch '1' on the modem is `on`. If you change the switch, unplug and power cycle the modem as explained above. If you get the message when `tip` is not

running, no one is dialed in, and there is no lock file, try unplugging the modem cable and/or power cycling the modem. Finally, you can do `ps ax` and look for a process tying up the port.

- If you get a `tip: /dev/cua0: Permission denied` or `link down` error message, make sure you have `:dv=/dev/cua0:` in `/etc/remote`. Check for a lock file in `/usr/spool` or `/usr/spool/uucp` called `LCK.*` where * is the name of the dial out destination. Make sure permission modes on `/dev/cua0` are 622, and it is owned by uucp. Try turning off the modem, unplugging it for a minute, and plugging it back in again. Also check permissions on the `/usr/spool/uucp` directory. It must exist and be readable, writable, and executable by everybody (777 mode).

5.5. Adding A Printer To Your System

The line printer spooling system implemented in Sun UNIX can handle a variety of printers and configurations. It handles local and remote printers, printers attached to serial lines, raster output devices such as Varian or Versatec, and laser printers such as an Imagen. It can handle multiple printers and multiple spooling queues.

Some printers require a serial interface, while others require a parallel interface; a printer can be hooked-up to a Sun Workstation in either of these two ways. You may connect to one of the serial ports on the workstation backpanel, or you may cable the printer directly to a Multibus printer controller board. Sun currently supports the Systech VPC-2200 Centronics/Versatec board. The difference between these two types of installation is great enough to merit a separate discussion for each. First we explain the addition of a printer to a serial port, followed by explanation of a parallel hook-up to the Systech VPC board. Further sections explain the operational aspects of the line printer system.

Hooking Up A Serial ASCII Printer

Many printers can be driven from an asynchronous serial port. They must be able to receive ASCII characters sent over a serial line. Connection to a serial port will save the cost of a Multibus board that drives your printer. If you are going to put a printer on one of the serial ports on your workstation, please read the section above called *Connecting Devices To Asynchronous Serial Ports*. The general discussion there will prepare you for the following installation instructions.

Serial Port and Cable Connection

We begin with the hardware configuration. Read the manufacturer's printer manual and check the following on the new printer.

- Determine whether the printer is configured as DTE (data terminal equipment) or DCE (data communication equipment). Set it to DTE if you can.
- If possible, disable the printer's use of modem control signals like CTS (clear-to-send) and DSR (data-set-ready). If the printer requires CTS and DSR, loop back the following lines on the printer: connect line 4 to line 5, and line 6 to line 20.
- If possible, enable xon/xoff flow control.
- At least for the installation phase, set the baud rate at a low rate for testing, 1200 for example.

Now make sure you have an available serial port on your workstation. The number of serial ports varies from one Sun model to another. Each Sun workstation provides at least two asynchronous serial ports controlled by the Serial Communications Controller on the Sun-2 CPU board. If all the existing serial ports are in use, and you are not planning to disconnect any current peripheral devices to make a port available, you will need to add a new Multibus board to the card cage to increase the number of serial ports. The instructions for adding a Multibus board are given above in the section *Adding A Board To Your System.*

When you do have a port available, connect the printer to the workstation with a three-line cable. If the printer is a DTE device, use a null-modem cable. Null modem cables have line 7 wired straight through and lines 2 and 3 swapped so

that proper transmit and receive signals are communicated between two DTE devices. If the printer is a DCE device, then connect 2, 3, and 7 straight through. Make sure all the connections are tight.

When you have connected the printer as explained above, you should verify that cabling and hardware are performing before proceeding. Obviously, if there is a problem with connection or faulty hardware, none of the later software installation will work.

- Consult the operation manual for the printer and run the printer's stand-alone diagnostics. If these do not run, call the printer manufacturer. Remember to set the switch on the printer back to operate mode after the self-test.
- Verify that the printer switch settings are correct. Check for the correct settings in the printer operation manual, and make sure they correspond with the parameters given above for the Sun.
- Send something over the tty line to the printer. To do this, set the characteristics of the printer using `stty` (1), and then `cat` (1) a file to the `/dev/tty*` serial port where the printer is attached. If the printer is attached to the first serial port on the CPU board, type:

  ```
  # (stty 1200; cat /etc/passwd) > /dev/ttya
  ```

  to print a copy of the password file.

  If the printer is 'dead', your cable may be bad, or you may need a null modem cable as described above.

  If something all garbled prints, `stty` may not have set all the terminal characteristics properly for the printer. Read the manufacturer's manual and check the switches on the printer. In particular check the baud rate, 1200 in this example. Make sure the printer and the `stty` setting match. You may have to set additional options with `stty` to match those on your printer. The most likely are: to set parity to `even`, `odd` or neither (`space` parity); to set tab expansion with `-tabs`; and to allow carriage return for newline, and output CR-LF for carriage return or newline with `-nl`. These parameters are set for the line printer spooling system software in the `/etc/printcap` file by the `fs` capability. See below for an explanation. Also see `printcap` (4) in *The System Interface Manual.*

File System Modification

You must be superuser to make the following file system modifications.

The Sun software includes the necessary programs to run the line printer system, and a default line printer queue is included. You need to create a `/etc/printcap` file to make the printer work properly.

Editing The `printcap` File

`printcap`(5) is a database describing printers. It describes line printers directly attached to a machine and printers accessible across a network. Each printer should have an entry, and it is a good idea to remove entries for printers not in your system. Typically, a system will have just a single printer. The syntax of entries in `printcap` can seem torturous; check carefully after you have modified the file. We give explanations and examples below.

`printcap` entries consist of fields that (except for the name field) **must** be preceded and followed by colons (:); the last field specified must terminate with a colon. The first field of each entry **must** be the name(s) the printer is known by, where these names are separated by '|' characters, and the field begins at the left margin without a leading colon. Every system **must** have a printer that uses `lp` as one of its aliases. In a single printer environment, you must include `lp` in the name field. Let's look at a sample `printcap` entry for a printer attached to a serial port, and then proceed with further explanations.

```
# printcap entry for DecWriter on serial port
0|lp|DEC|decwriter|LA-180 DecWriter III:\
    :lp=/dev/ttya:sd=/usr/spool/lpd:br#1200:fs#06020:fc#0300:\
    :tr=\f:xs#040:of=/usr/lib/lpf:lf=/usr/adm/lpd.errs:
```

We can learn several general things from this example. First, we see that comments are allowed if a line begins with a '#'. Next something not so obvious to see, an entry is continued onto another line with the '\' character followed, without blank space, by a carriage return. This is a **must**. Blank space accidently typed at the end of a `printcap` line will cause severe problems. We notice that the continuation lines of this example are tabbed over from the left margin; all continuation lines in an entry **must** begin with blank space, normally a tab character. When entries do continue onto second and subsequent lines, a colon **must** appear at the end of one line and the beginning of the next.

Now, let's turn to the fields within each entry. The `printcap` manual page gives a table of all the capabilities available. You will probably need just the ones shown here to run most line printers from a serial port.

As mentioned above, the first field of each entry gives the names the printer is known by. One of the names must be `lp`; on a machine with more than one printer, one printer must use `lp` as one of its names — but only one printer.

Notice that each subsequent field is introduced by a two character code. Numeric capabilities take the form: *character_code#number_value*; for example, `br#1200`. String capabilities take the form: *character_code=sequence*; for example, `lf=/usr/adm/lpd-errs`. The capabilities shown in this entry are:

- `lp` – Specifies the file name to be opened for output. In this case it is the first serial port `/dev/ttya`. The default is `/dev/lp`.
- `sd` – Specifies the name of a spooling directory. Make sure the directory exists with proper permissions before attempting to run the printer. When you install UNIX, it includes a correct spool directory: `/usr/spool/lpd`.
- `br` – Sets the baud rate for the tty line to the value given here.
- `fs` – Sets flag bits. See `tty(4)` for an explanation of these `sg_flags`. The example here, `fs#06020`, is a good one to use on a serial printer. The 6000 value expands tabs, that is, sends blank spaces to a printer that cannot use a tab character. The 20 value puts out both a carriage return and a line-feed at the end of each line of print. Use the `fc` capability to clear flag bits.

- `fs` – Clears flag bits. Similar to `fs`, but turns off bits instead of turning them on. The `fc#0300` turns off the even and odd parity bits, resulting in space parity.
- `tr` – A trailer string, to be sent to the printer at the end of a series of print jobs. In this example the trailer is a form feed.
- `xs` – Sets local mode bits. If you want 8 bits out, no parity, set `xs` to 40. Use the `xc` capability to clear local mode bits. Setting `litout`, however, disables regular output processing such as tab expansion and CR/LF processing.
- `of` – This is the name of an output filtering program, a standard post-processor for nroff in this example. It is supposed to make underlined text come out properly on line printers. It may or may not be appropriate on your printer. Try running your printer without it. If the format looks fine, you probably do not need the filter.
- `lf` – This is the name of the file where spooler errors will be logged. It can be created anywhere, but must exist with write permissions before errors can be sent to it. To assure that the spooler daemon can write on the error log file, become superuser, create the log file, and do a `chmod 666` on it.

Other File System Modifications

After your `printcap` file has been edited for your printer(s), you will need to make a few more adjustments to the UNIX file system.

- Check to make sure the proper permissions and ownerships exist on the files `/usr/lib/lpd`, `/usr/ucb/lpr`, and on the directory, `/usr/spool/lpd`. Note that you may have given a different name to your `/usr/spool/lpd` directory, and that you will have one spooling directory with a unique name for each printer accessible on your system. For the files named above type `ls -lg` to check permissions, ownership and group. To check the same things on the spooling directory, type `ls -lgd`. In addition, check the files in the spooling directory with `ls -lg`. The permissions, ownership, and group should match those shown below:

```
# ls -lg /usr/lib/lpd /usr/ucb/lpr
-rws--s--x  1 root    daemon   53248 Oct 14 09:19 /usr/lib/lpd
-rws--s--x  1 root    daemon   30720 Oct 14 09:19 /usr/ucb/lpr
# ls -lgd /usr/spool/lpd
drwxrwx---  2 daemon  daemon     512 Nov 09 11:00 /usr/spool/lpd
# ls -lg /usr/spool/lpd
-rw-r--r--  1 root     daemon         22 Mar  1 18:25 lock
-rw-rw-r--  1 root     daemon         29 Mar  1 17:28 status
```

- Make sure that `init`(8) does not create a login process on the port you are using for your printer. To do this, edit the `/etc/ttys` file, putting a zero (0) in the first column of the entry for the tty port that your printer is attached to. The example `printcap` attaches a serial printer to the first CPU serial port, `ttya`; `/etc/ttys` has an entry like:

```
02ttya
```

If it was necessary to edit the `/etc/ttys` file, you must notify `init` to

make the system aware of the changes made. That is done by the following `kill` command while you are superuser:

```
# kill -1 1
```

Now you can send something to the printer with the `lpr` command. If the printer does not work now, but did when you sent it output with the `cat` command to `/dev/tty*`, make sure the permissions and characteristics of all the `lpr`–related files are correct. They must be as described here. Make sure that your output filter (if you have one) is executable and that it is located where `/etc/printcap` says it is. Finally, check all the fields and the syntax in the `/etc/printcap` file.

Hooking Up A Printer To A VPC-2200 Multibus Board

In this configuration, you hook-up your printer to a Multibus printer controller. Sun currently provides support in the standard software distribution for a single Systech VPC-2200 board per workstation. We explain the installation of that board and a printer in this section.

If you want to use a different printer controller board you will have to write your own device driver for the kernel. This requires some expertise; the procedure is described in *Writing Device Drivers For The Sun Workstation* in the *System Internals Manual*. Due to the difficulty of writing a driver, we do not recommend this course.

The remaining discussion deals with the Systech VPC-2200 board.

First install the board in the card cage of your Sun Workstation and connect the cable. If necessary, configure the new board into the system kernel. Finally modify the UNIX file system to enable the workstation to queue jobs and send them to the printer.

These steps are discussed in the sections below.

Card Cage Installation And Cable Hook-up

Read the manufacturer's manual for the VPC-2200. Check all recommendations and settings carefully. When installing a printer controller board, you may place it in any slot that does not share a P2 section with the Sun-2 CPU or Sun-2 Memory boards. A basic ribbon cable can connect the board to the printer.

You should note the section *Systech VPC-2200 Versatec Printer/Plotter Controller* in the *Hardware Manual* for your workstation. There you will find a detailed description of how to install and test the Systech board.

Follow the self test instructions for the board once it is installed. (Note that some printers, such as the Imagen, will not work with self test.) Remember to set switches back to operate mode after self test is complete. If you have trouble with self test, double check all the recommended switch settings, check the cabling from the controller to the printer, and make sure the board is connected snugly in the card cage. Consult the manufacturer's manual for any recommended procedures. Call the manufacturer if the board still seems flaky.

Kernel Modification

You must be superuser to make kernel modifications.

Here we give a brief outline of the steps for building a new kernel after installing a Systech VPC-2200 printer interface board. For a general discussion of software modifications after adding a new board, see the section *Adding A Board To Your System* in this manual. (If you use a different board and write your own device driver, the process will be slightly different; see the section *Writing Device Drivers For The Sun Workstation* in the *System Internals Manual.*)

Follow the steps shown here to reconfigure the kernel after the installation of a Systech printer controller board. These steps explain the process on a system named GRENDEL:

1) Change to `/sys/conf` directory and make a copy of the current kernel configuration file to keep until the new one is installed and running.

```
# cd /sys/conf
# cp GRENDEL old.GRENDEL
```

2) Edit the kernel configuration file, GRENDEL in this example, adding an entry for the Systech board you have installed into the card cage. Look in *Section 4* of the *System Interface Manual* for a description of this device. The entry you make in the kernel configuration file looks like this:

```
device vpc0 at mb0 csr 0x480 priority 2
```

3) Run `/etc/config` on the new kernel configuration file.

```
# /etc/config GRENDEL
```

4) Build the new kernel.

```
# cd ../GRENDEL
# make depend
# make
```

5) Install the new kernel and try it out.

```
# cp vmunix /newvmunix
# /etc/halt
> b newvmunix -s
```

6) If the new kernel appears to work, save the old kernel and install the new one in `/vmunix`.

```
# cd /
# mv vmunix ovmunix
# mv newvmunix vmunix
# /etc/reboot
```

Remember to remove the `old.GRENDEL` kernel configuration file you created as a backup.

For more information about kernel building, see the manual *Installing UNIX On The Sun Workstation.*

File System Modification

You must be superuser to make the following system modifications.

Using MAKEDEV To Create Special Files

The UNIX kernel communicates with the printer through special files in the directory `/dev`. When a Systech board is added to the system, these new entries must be made in the `/dev` directory. This is relatively easy to do with a shell script called `MAKEDEV` (8), located in `/dev`. `MAKEDEV` takes the argument `vpc0`, and automatically installs the files `/dev/vp0` and `/dev/lp0`. For example:

```
# cd /dev
# MAKEDEV vpc0
```

is what you type. If the system has had a printer in the past, there may be a `/dev/vp0`, or even a `/dev/lp0`, existing already. If they exist, you **should** remove these files before running `MAKEDEV`. An error message from `MAKEDEV`, `mknod failed`, will be returned if you try to make a file on an existing one.

Editing The /etc/printcap File

`printcap`(5) is a database describing printers. It describes line printers directly attached to a machine and printers accessible across a network. Each printer should have an entry, and it is a good idea to remove entries for printers not in your system. Typically, a system will have just a single printer. The syntax of entries in `printcap` can seem torturous, check carefully after you have modified the file. We give explanations and examples below.

`printcap` entries consist of fields that (except for name fields) **must** be preceded and followed by colons (:); the last field specified must terminate with a colon. The first field of each entry **must** be the name(s) the printer is known by, where these names are separated by bar (|) characters, and the field begins at the left margin without a leading colon. Every system **must** have a printer that uses `lp` as one of its aliases. In a single printer environment, you must include `lp` in the name field. Let's look at a sample `printcap` entry for a Versatec printer attached to a Systech board, and then proceed with further explanations.

```
# printcap entry for Versatec V80
0|lp|V80|versatec:\
     :lp=/dev/vp0:sd=/usr/spool/lpd:pl#66:px#2112:py#1700:tr=\f:\
     :of=/usr/lib/vpf:if=/usr/lib/vpf:tf=/usr/lib/rvcat:\
     :cf=/usr/lib/vdump:vf=/usr/lib/vpltdmp:lf=/usr/spool/lpd-errs:
```

We can learn several general things from this example. First, we see that comments are allowed if a line begins with a '#'. Next something not so obvious to see, an entry is continued onto another line with the '\' character followed, without blank space, by a carriage return. This is a **must**. Blank space accidently typed at the end of a `printcap` line will cause severe problems. We notice that the continuation lines of this example are tabbed over from the left margin; all continuation lines in an entry **must** begin with blank space, typically a tab. When entries do continue onto second and subsequent lines, a colon **must** appear at the end of one line and the beginning of the next.

Now, let's turn to the fields within each entry. The `printcap` manual page gives a table of all the capabilities available. Those shown here will run a Versatec 80 printer.

As mentioned above, the first field of each entry gives the names the printer is known by. One of the names must be `lp`; on a machine with more than one printer, one printer must use `lp` as one of its names — but only one printer.

Notice that each subsequent field is introduced by a two character code. Numeric capabilities take the form: *character_code#number_value*; for example, `px#2112`. String capabilities take the form: *character_code=sequence*; for example, `lf=/usr/spool/lpd-errs`. The capabilities shown in this entry are:

- `lp` – Specifies the device name to be opened for output.
- `sd` – Specifies the name of a spooling directory. Make sure the directory exists with proper permissions before attempting to run the printer.
- `pl` – Sets the page length in lines.
- `px` – Sets the page width in pixels.
- `py` – Sets the page length in pixels.
- `tr` – Specifies a trailer character, in this case a form feed, to be sent to the printer at the end of a series of print jobs, making the paper easier to remove.
- `of` – This is the name of an output filtering program to set up the data for the Versatec printer. When `if` is specified, `of` is used only for the banner page. See the section below *Output Filters*.
- `if` – This is the name of an output filtering program which can do accounting.
- `tf` – A troff output filter for the Versatec printer.
- `cf` – A plot output filter for the Versatec printer.
- `vf` – A raster image and screen dump filter for the Versatec printer.
- `lf` – This is the name of the file where spooler errors will be logged. It can be created anywhere, but must exist with write permissions before errors can be sent to it. To assure that the spooler daemon can write on the error log file, become superuser, create the file, and do a `chmod 666` on it.

Other File System Modifications

After your `printcap` file has been edited for your printer(s), you will need to make a few more adjustments to the UNIX file system.

- Check to make sure the proper permissions and ownerships exist on the files `/usr/lib/lpd`, `/usr/ucb/lpr`, and on the directory, `/usr/spool/lpd`. Note that you may have given a different name to your `/usr/spool/lpd` directory, and that you will have one spooling directory with a unique name for each printer accessible on your system.

For the files named above type `ls -lg` to check permissions, ownership and group. To check the same things on the spooling directory, type `ls -lgd`. In addition, check the files in the spooling directory with `ls -lg`. The permissions, ownership, and group should match those shown below, for example:

```
# ls -lg /usr/lib/lpd /usr/ucb/lpr
-rws--s--x  1 root    daemon   53248 Oct 14 09:19 /usr/lib/lpd
-rws--s--x  1 root    daemon   30720 Oct 14 09:19 /usr/ucb/lpr
# ls -lgd /usr/spool/lpd
drwxrwx---  2 daemon  daemon     512 Nov 09 11:00 /usr/spool/lpd
# ls -lg /usr/spool/lpd
-rw-r--r--  1 root     daemon         22 Mar  1 18:25 lock
-rw-rw-r--  1 root     daemon         29 Mar  1 17:28 status
```

Now you can send something to the printer with the `lpr` command. If the printer does not work now, make sure the permissions and characteristics of all the `lpr`–related files are correct. They must be as described here. Make sure that your output filter (if you have one) is executable and that it is located where `/etc/printcap` says it is. Finally, check all the fields and the syntax in the `/etc/printcap` file.

Note that an Imagen printer comes with its own software and installation instructions. These are similar to the above, but not identical. Follow the Imagen instructions.

Printing On Remote Machines

Remote spooling via the network is handled with two spooling queues, one on the local machine and one on the remote machine. When a remote printer job is initiated with `lpr`, the job is queued locally and a daemon process is created to oversee the transfer of the job to the remote machine. If the destination machine is unreachable, the job will remain queued until it is possible to transfer the files to the spooling queue on the remote machine.

The remote (or receiving) machine must have an entry for the local (or sending) machine in both its `/etc/hosts` and its `/etc/hosts.equiv` files to enable the transfer.

The local machine must know about the remote machine by having an entry for it (remote) in the local's `/etc/hosts` file.

Machines which use the printer on a remote machine should have an empty field for the `lp` capability in the `/etc/printcap` file. For example, the following `printcap` entry would send output to the printer named 'versatec' on the remote machine 'venus'.

```
lp|1|versatec:\
    :lp=:rm=venus:rp=versatec:sd=/usr/spool/vpd:
```

- The name field in this example happens to match the name of the printer on the remote machine. It need not necessarily.
- `lp` – is an empty field. Don't forget the equal sign.
- `rm` – is the name of the remote machine to print on. As mentioned, this name must appear in the `/etc/hosts` database on the local machine.

- `rp` – indicates the name of the printer on the remote machine is 'versatec'; the default for this capability is the 'lp' printer on remote.
- `sd` – specifies `/usr/spool/vpd` is the spooling directory instead of the default value of `/usr/spool/lpd`.

Output Filters

The `printcap` examples above show various types of output filters. This section gives more details on their uses and specifications.

Filters are used to handle device dependencies and to perform accounting functions. The output filter **of** is used to filter text data to the printer device when accounting is not used or when all text data must be passed through a filter. It is not intended to perform accounting since it is started only once, all text files are filtered through it, and no provision is made for passing owners' login name, identifying the beginning and ending of jobs, etc. The other filters, such as `if`, (if specified) are started for each job printed and perform accounting if there is an `af` entry. The `af` entry designates the file where `if` puts its accounting informations — see the second example below. If entries for both `of` and one of the other filters are specified, the output filter is used only to print the banner page; it is then stopped to allow other filters access to the printer. An example of a printer which requires output filters is the Benson-Varian.

```
va|varian|Benson-Varian:\
     :lp=/dev/va0:sd=/usr/spool/vad:of=/usr/lib/vpf:\
     :tf=/usr/lib/rvcat:mx#2000:pl#66:tr=\f:
```

The `tf` filter (invoked with `lpr -t`) takes a `troff` output file and converts it to Versatec output. It is used by `vtroff` (1). Note that the page length is set to 66 lines by the `pl` entry for 8.5" by 11" fan-fold paper. To enable accounting, the varian entry would be augmented with an `af` file as shown below.

```
va|varian|Benson-Varian:\
     :lp=/dev/va0:sd=/usr/spool/vad:of=/usr/lib/vpf:\
     :if=/usr/lib/vpf:tf=/usr/lib/rvcat:af=/usr/adm/vaacct:\
     :mx#2000:pl#58:tr=\f:
```

Output Filter Specifications

Sun software provides several filters which are listed as `?f` under CAPABILITIES in `printcap`(5). For many devices or accounting methods, it is probably necessary to create a new filter.

Filters are spawned by `lpd` with their standard input the data to be printed, and standard output the printer. The standard error is attached to the `lf` file for logging errors. A filter must return a '0' exit code if there were no errors, '1' if the job should be reprinted, and '2' if the job should be thrown away. When `lprm` sends a kill signal to the `lpd` process controlling printing, it sends a SIGINT signal to all filters and descendents of filters. This signal can be trapped by filters which need to perform cleanup operations such as deleting temporary files.

Arguments passed to a filter depend on its type. The `of` filter is called with the following arguments.

ofilter –w*width* –l*length*

The *width* and *length* values come from the `pw` and `pl` entries in the printcap database. The `if` filter is passed the following parameters.

filter [**-c**] **-w***width* **-l***length* **-i***indent* **-n** *login* **-h** *host accounting_file*

The -c flag is optional, and only supplied when control characters are to be passed uninterpreted to the printer (when the -l option of `lpr` is used to print the file). The -w and -l parameters are the same as for the `of` filter. The -n and -h parameters specify the login name and host name of the job owner. The last argument is the name of the accounting file from `printcap`.

All other filters are called with the following arguments:

filter **-x***width* **-y***length* **-n** *login* **-h** *host accounting_file*

The -x and -y options specify the horizontal and vertical page size in pixels (from the `px` and `py` entries in the printcap file). The rest of the arguments are the same as for the `if` filter.

Line Printer Commands

The four sections below discuss the system commands that run the printer. A short description is given here, read the appropriate manual page for further details.

lpd – The Printer Daemon

The program `lpd` (8), usually invoked at boot time from the `/etc/rc` file, acts as a master server for coordinating and controlling the spooling queues configured in the printcap file. When `lpd` is started it makes a single pass through the `printcap` database, restarting any printers which have jobs. In normal operation `lpd` listens for service requests on multiple sockets, one in the UNIX domain (named `/dev/printer`) for local requests, and one in the Internet domain (under the 'printer' service specification) for requests for printer access from off machine; see `socket` (2) and `services` (5) for more information on sockets and service specifications. `Lpd` spawns a copy of itself to process the request; the master daemon continues to listen for new requests.

Client processes communicate with `lpd` using a simple transaction oriented protocol. Authentication of remote clients is done based on the 'privilege port' scheme employed by `rshd` (8C) and `rcmd` (3X). The following table shows the requests understood by `lpd`. In each request the first byte indicates the 'meaning' of the request, followed by the name of the printer to which it should be applied. Additional lines containing qualifiers may follow, depending on the request.

Request	Interpretation
^A*printer*	check the queue for jobs and print any found
^B*printer*	receive and queue a job from another machine
^C*printer* [*users* ...] [*jobs* ...]	return short list of current queue state
^D*printer* [*users* ...] [*jobs* ...]	return long list of current queue state
^E*printer person* [*users* ...] [*jobs* ...]	remove jobs from a queue

lpc — Line Printer Control Program

The `lpc` (8) program is used by the system administrator to control the operation of the line printer system. For each line printer configured in `/etc/printcap`, `lpc` may be used to:

- disable or enable a printer,
- disable or enable a printer's spooling queue,
- rearrange the order of jobs in a spooling queue,
- find the status of printers, and their associated spooling queues and printer dameons.

The major commands and their intended use are described here. The command format and remaining commands are described in `lpc` (8).

`abort` and `start`

`Abort` terminates an active spooling daemon on the local host immediately and then disables printing (preventing new daemons from being started by `lpr`). This is normally used to force a hung line printer daemon to restart (i.e., `lpq` reports that there is a daemon present but nothing is happening). It does not remove any jobs from the queue (use the `lprm` command instead). `Start` enables printing and requests `lpd` to start printing jobs.

`enable` and `disable`

`Enable` and `disable` allow spooling in the local queue to be turned on/off. This will allow/prevent `lpr` from putting new jobs in the spool queue. It is frequently convenient to turn spooling off while testing new line printer filters since the `root` user can still use `lpr` to put jobs in the queue but no one else can. The other main use is to prevent users from putting jobs in the queue when the printer is expected to be unavailable for a long time.

`restart`

`Restart` allows ordinary users to restart printer daemons when `lpq` reports that there is no daemon present.

`stop`

`Stop` is used to halt a spooling daemon after the current job completes; this also disables printing. This is a clean way to shutdown a printer in order to perform maintenance, etc. Note that users can still enter jobs in a spool queue while a printer is stopped.

`topq`

`Topq` places jobs at the top of a printer queue. This can be used to reorder high priority jobs since `lpr` only provides first-come-first-serve ordering of jobs.

lpr — Enter Jobs Into Print Queue

The `lpr` (1) command is used by users to enter a print job in a local queue and to notify the local `lpd` that there are new jobs in the spooling area. `Lpd` either schedules the job to be printed locally, or in the case of remote printing, attempts to forward the job to the appropriate machine. If the printer cannot be opened or the destination machine is unreachable, the job will remain queued until it is possible to complete the work.

For each file you want to print, `lpr` creates a control file and a separate data file in the spool directory. The control file is named `cf`*XnnnhostnameX*, and the data file is named `df`*XnnnhostnameX*. The fields in the names have the following meanings:

- `cf` — control file
- `df` — data file
- *X* — alpha character A, B, C, etc.
- *nnn* — numeric value taken from the .seq file in the spool directory
- *hostname* — hostname of machine from which the job originated.

lpq — Show Line Printer Queue

The `lpq` (1) program works recursively backwards displaying the queue of the machine with the printer and then the queue(s) of the machine(s) that lead to it. `Lpq` has two forms of output: in the default, short, format it gives a single line of output per queued job; in the long format it shows the list of files, and their sizes, which comprise a job.

lprm — Remove Jobs From A Queue

The `lprm` (1) command deletes jobs from a spooling queue. If necessary, `lprm` will first kill off a running daemon which is servicing the queue, restarting it after the required files are removed. When removing jobs destined for a remote printer, `lprm` acts similarly to `lpq` except it first checks locally for jobs to remove and then tries to remove files in queues off-machine.

Access Control

The printer system maintains protected spooling areas so that users cannot circumvent printer accounting or remove files other than their own. The strategy used to maintain protected spooling areas is as follows:

- The spooling area is writable only by a `daemon` user and `daemon` group.
- The `lpr` program runs setuid `root` and setgid `daemon`. The `root` access is used to read any file required, verifying accessibility with an `access` (2) call. The group ID is used in setting up proper ownership of files in the spooling area for `lprm`. Data files are owned by user, group daemon, mode 660.
- Control files in a spooling area are made with `daemon` ownership and group ownership `daemon`. Their mode is 0660. This insures control files are not modified by a user and that no user can remove files except through `lprm`.
- The spooling programs, `lpd`, `lpq`, and `lprm` run with setuid `root` and setgid `daemon` to access spool files and printers.

- The printer server, `lpd`, uses the same verification procedures as `rshd` (8C) in authenticating remote clients. The host on which a client resides must be present in the file /etc/hosts.equiv, used to create clusters of machines under a single administration.

In practice, none of `lpd`, `lpq`, or `lprm` would have to run as user `root` if remote spooling were not supported. In previous incarnations of the printer system `lpd` ran setuid `daemon`, setgid `spooling`, and `lpq` and `lprm` ran setgid `spooling`.

Problems

If you have problems after installation, check the cabling first, as outlined above in the section *Connecting Devices To Asynchronous Serial Ports*.

For serial printers, check the information in the `/etc/ttys` file, and make sure you have used `kill` to signal `init` as explained above.

If the printer does not work or behaves strangely, check the `/etc/printcap` file. You may have unexpected whitespace, or you may have forgotten to escape the newline character in your entry: a backslash (\) followed immediately by a newline (carriage return) will continue that line on the next. If there any output filters specified in the `printcap` entry, they could be the source of problems; make sure they have not introduced characters the printer does not know about.

Error Messages From The Line Printer System

There are a number of messages which may be generated by the line printer system. This section categorizes the most common and explains the cause for their generation. The messages are grouped under the command which generates them. Where the message indicates a failure, directions are given to remedy the problem.

In the examples below, the name `printer` is the name of the printer. This would be one of the names from the `printcap` database.

lpr

`lpr: `*printer*`: unknown printer`

The `printer` specified by the `-P` option or the `lp` default, was not found in the `printcap` database. Usually this is a typing mistake; however, it may indicate a missing or incorrect entry in the /etc/printcap file.

`lpr: `*printer*`: jobs queued, but cannot start daemon.`

The connection to `lpd` on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket /dev/printer to be sure it still exists (if it does not exist, there is no `lpd` process running). Use

```
# ps ax | fgrep lpd
```

to get a list of process identifiers of running lpd's. The `lpd` to kill is the one which is not listed in any of the 'lock' files (the lock file is contained in the spool directory of each printer). Kill the master daemon using the following command.

```
# kill pid
```

Then remove `/dev/printer` and restart the daemon (and printer) with the following commands.

```
# rm /dev/printer
# /usr/lib/lpd
```

`lpr: `*printer*`: printer queue is disabled`

This means the queue was turned off with

```
# lpc disable printer
```

to prevent `lpr` from putting files in the queue. This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by a super-user with `lpc`.

lpq

`waiting for `*printer*` to become ready (offline ?)`

The printer device could not be opened by the daemon. This can happen for a number of reasons, the most common being that the printer is turned off-line. This message can also be generated if the printer is out of paper, the paper is jammed, etc. The actual reason is dependent on the meaning of error codes returned by system device driver. Not all printers supply sufficient information to distinguish when a printer is off-line or having trouble (e.g. a printer connected through a serial line). Another possible cause of this message is some other process, such as an output filter, has an exclusive open on the device. Your only recourse here is to kill off the offending program(s) and restart the printer with `lpc`.

printer `is ready and printing`

The `lpq` program checks to see if a daemon process exists for `printer` and prints the file `status`. If the daemon is hung, a super user can use `lpc` to abort the current daemon and start a new one.

`waiting for `*host*` to come up`

This indicates there is a daemon trying to connect to the remote machine named *host* in order to send the files in the local queue. If the remote machine is up, `lpd` on the remote machine is probably dead or hung and should be restarted as mentioned for `lpr`.

`sending to `*host*

The files should be in the process of being transferred to the remote `host`. If not, the local daemon should be aborted and started with `lpc`.

`Warning: `*printer*` is down`

The printer has been marked as being unavailable with `lpc`.

`Warning: no daemon present`

The `lpd` process overseeing the spooling queue, as indicated in the 'lock' file in that directory, does not exist. This normally occurs only when the daemon or output filter has unexpectedly died. The error log file for the printer should be checked for a diagnostic from the deceased process. To restart an `lpd`, use

 # lpc restart *printer*

lprm

`lprm:` *printer* `: cannot restart printer daemon`

This case is the same as when `lpr` prints that the daemon cannot be started.

lpc

`couldn't start printer`

This case is the same as when `lpr` reports that the daemon cannot be started.

`cannot examine spool directory`

Error messages beginning with `cannot ...` are usually due to incorrect ownership and/or protection mode of the lock file, spooling directory or the `lpc` program.

lpd

The `lpd` program can write many different messages to the error log file (the file specified in the `lf` entry in `printcap`). Most of these messages are about files which can not be opened and usually indicate the `printcap` file or the protection modes of the files are not correct. Files may also be inaccessible if people manually manipulate the line printer system (i.e. they bypass the `lpr` program).

In addition to messages generated by `lpd`, any of the filters that `lpd` spawns may also log messages to this file.

6

Periodic Maintenance

Periodic Maintenance 177

6.1. Backing Up The File System With dump 178
6.2. Bootstrap And Shutdown Procedures 180
Powering-up Self Test Procedures 180
The Automatic Boot Procedure 182
Booting From Specific Devices 183
Shutdown Procedures 186
Power Loss 186
Messages from the Monitor and the Boot Program 187
6.3. When The System Crashes 197
Crash Error Messages 197
System Core Dumps 197
Analyzing System Core Dumps 199
User Program Crashes 199
When To Call For Help 200
6.4. Kernel Configuration 201
Notes to Step 3 of Kernel Configuration 201
Rules for Defaulting System Devices 203
Configuration File Grammar 204
Data Structure Sizing Rules 207
6.5. System Log Configuration 209
6.6. Monitoring System Performance 210
6.7. Resource Control 211

6.8. Accounting 212
6.9. Making Local Modifications 213
6.10. Files Needing Periodic Attention 214

6

Periodic Maintenance

This chapter discusses many of the tasks that the system administrator will have to perform. Some of these things will happen according to the schedule you make. Others are infrequent, even random, but they will have to be done eventually.

First we talk about the need for backing up the file system. Then we cover booting and shutting down the system. Next, a large section explains what to do when your system crashes. There are short sections explaining the system log configuration and system performance. Following that are some remarks about accounting, local modifications on your system, and files that need periodic attention.

6.1. Backing Up The File System With dump

It is extremely important that someone be responsible for making regularly scheduled backups of all your UNIX file systems. If something unusual happens to the system, any file could become lost or corrupted. When a file gets lost, you can only restore a version as recent as your last backup copy.

The dump (8) command allows you to copy file systems to tape for backup and preservation. You can backup all the files on a file system with a 'level 0' dump. The 'dump level' argument allows you to make incremental dumps at levels other than zero — only some of the file system's files are written to tape. Files are written to an incremental (non zero) level dump tape depending on when they were last modified. On the chosen file system, all files modified since a lower level dump are written to the dump tape. For instance, a level 0 dump writes all of a file system's files to tape, no matter when they were last modified; a level x (non zero) writes only those files modified since the last level y of lower value than x. Valid dump level arguments are 0-9. Level 0's and level 9's should be used by every system. It is not practical to execute a level 0 dump each day. The vast majority of the files on most file systems are not changed daily, and you would waste time and storage tapes.

Here are some hints to help you organize your backup procedure:

- You should do incremental dumps every working day, commonly level 9. You will save all the files modified that day, as well as those files still on the system that have been modified since the last dump of a level lower than 9.
- As you continue to do level 9's each day, the dump will grow larger, reflecting the growing number of files changed on the file system. If you then do a level 0 once a week you will save the entire system with the changes from the week.
- Of course, a file changed on Tuesday and then again on Thursday, will go onto Friday's level 0 dump looking like it did Thursday night, not Tuesday night. If you need to see the Tuesday version you are out of luck unless you have a Tuesday dump tape. (Or a Wednesday dump tape for that matter.) Similarly, a file present on Tuesday and Wednesday, but removed on Thursday, will not appear on the Friday level 0.
- We recommend that you do a level 9 each day the system is used, and that you save a week's worth of level 9 dumps for at least one week after they have been made.
- Once a week is probably a good interval to do some level of incremental dump which you save for a longer period.
- These procedures require an ample supply of tapes, some physical organization, and a reasonable schedule.

You should probably do a level 0 dump of your root file system once a week. Depending on the size of other file systems you may want to do incremental dumps (levels 3, 5, 7 for example) on them rather than level 0's. You could then do the level 0 dumps at longer intervals, provided you saved all the lowest level incremental tapes between the level 0 dumps.

- For example you could do level 9's daily and keep the daily tapes for one week.
- Once a week you could do a level 5 and keep the weekly tapes for a month.
- Once a month you could do a level 3 and keep the monthly tapes for one year.

In this way you would not **need** to do a level 0 for one year, yet would still have all the modified files backed up. This is simply an example; **we recommend** doing level 0's more frequently. Remember to keep them, and the incremental tapes, around for a good long time. There is no telling when someone will come to the system administrator and ask for a copy of a file that existed for one week back in the beginning of the year.

Each site will decide on the back-up schedule and frequency that work best for it. The most important thing is to settle upon some schedule and make sure the dump tapes are made. In addition, make sure stored tapes are kept cool and clean and are well labeled. Tapes that cannot be read due to deterioration or lack of a label are useless.

For an explanation of how to back up and restore files, see the section *Backing Up And Restoring File Systems* in Chapter 2 of this manual.

6.2. Bootstrap And Shutdown Procedures

This section discusses starting and stopping the UNIX operating system on a Sun workstation.

- First we discuss powering-up the workstation, and the monitor that controls the system before the UNIX kernel has taken control.
- From the monitor there are several ways of bootstrapping or "booting up" the UNIX kernel, we will give some guidelines for deciding how to choose among the various boot procedures when the automatic boot procedure has been interrupted.
- Next, we discuss safe ways to stop the UNIX system, and what to do if there is a power loss during system operation.
- Finally we list the messages generated by the monitor and the Boot program. They can be helpful in debugging problems.

The most important part of starting and stopping the UNIX operating system is to preserve the integrity of its file system!

Powering-up Self Test Procedures

The central processor board (CPU) of the Sun Workstation has a set of PROMs which contains a program generally known as the 'monitor'. The monitor controls the operation of the system before the UNIX kernel takes control.

Under normal circumstances, the monitor automatically bootstraps the UNIX system after initial power-on; no user intervention is required.

When system power is first turned on, the monitor runs a quick self-test procedure. The test can have one of these results:

- Critical errors are found. The screen remains dark. The error is reported on eight LEDs on the CPU board in the card cage.
- No video board is found. The monitor sends its output to serial port 'A' (labeled 'RS-232 A' on Model 100U/150U backpanels, 'SIO-A' on Model 120/170 backpanels, and 'SIO-A' or 'UART-A' or 'Serial Port A' on Models 50 and 160). Connect an ASCII terminal to this RS-232 connector. (Sun supports two terminals, the TVI 925, and the Wyse 50 in TVI 925 mode.) Configure the terminal for 7 bits, even parity, flow control enabled, 9600 baud. Then power on the workstation again and look for messages on the terminal.
- Non-critical errors are found. These are reported to the screen, and the system begins the automatic boot process.
- No errors are found. This is reported to the screen, and the system begins the automatic boot process.

When Critical Errors Are Found In Self Test

Severe problems are reported on the eight small LEDs on the CPU board. Depending on your workstation model, you get at these differently:

- With a Sun-1/100U, these LEDs can be glimpsed through the cooling slots on the left side of the monitor base. If your screen remains dark, and you see more than one LED on in the middle of some board in the card cage, you'll probably have to slide the drawer out of your Sun-1/100U to really see the

error code in the LEDs.

- With a Sun-1/150U or Sun-2/170, you can see the LEDs by opening the front door of the cardcage enclosure.
- With a Sun-2/120, pull off the front plastic panel of the pedestal; the row of 8 LEDs is on the front edge of the CPU card in the far left slot of the cardcage.
- With a Sun-2/50, the LEDs are marked 'Diag Leds' at the left edge of the panel covering the CPU board.
- With a Sun-2/160, the LEDs are at the top edge of the leftmost (the CPU) board stacked vertically in the rack.

When power is first applied to the workstation, all eight LEDs light, then each lights quickly in sequence. Following this 'lamp test', the lights blink rapidly as each test is passed. The lights slow down as memory is tested; each of the two memory tests takes a few seconds per megabyte. Finally, three LEDs on the end light momentarily, then all the LEDs go off except for a middle LED which blinks about once a second. If your workstation follows this sequence, self-test has not found a critical problem. (Once UNIX or other programs have gained control of the system, they can use the LEDs in other ways. This description only applies to the power-on sequence.)

If at some point in the above sequence, the lights freeze (keep the same pattern for more than a minute), or the sequence restarts from the beginning, there is a critical hardware problem with the workstation. The appropriate thing to do in this case is to contact Sun Microsystems Field Service or your local Field Service organization. Copy down the pattern of lights (as well as you can, if it is repeating over and over); they contain important diagnostic information for Field Service.

When Non-Critical Errors Are Found In Self Test

Non-critical errors result in a display like the following:

```
Self Test found a problem in something

Wrote wdata at address addr, but read rdata.
Damage found, damages

--> Give the above information to your service-person

Sun Workstation, Model model_number , type_of_keyboard .
ROM Rev N, some_number_MB memory installed
Serial #some_number, Ethernet address n:n:n:n:n:n

Auto-boot in progress . . .
```

something shows what part of the system was most recently found to be malfunctioning. (If more than one error occurs, a summary of all errors is given in the *damages* section, and details about the last error are reported here.)

wdata is the data that was written into part of the system, or which was expected to be there if the system was functioning normally.

addr is the address where the data was read and/or written. For memory errors, this is a physical memory address; for other errors, the interpretation of the address depends on *something*.

rdata is the data that was read back from *addr* and was found to be invalid because it was not the same as *wdata*.

damages is a list of all subsystems which were found to have errors. There is not enough room to save information about all of the errors that were found (only the last one), but this minimal information about each is recorded.

You should copy down the information from the screen, then call Sun Microsystems Field Service, or your local Field Service representative. The system will attempt to bootstrap itself despite the error.

When No Errors Are Found In Self Test

The following display results:

```
Self Test completed successfully.

Sun Workstation, Model model_number, type_of_keyboard.
ROM Rev N, some_number_MB memory installed
Serial #some_number, Ethernet address n:n:n:n:n:n

Auto-boot in progress . . .
```

The monitor then begins auto-boot.

The Automatic Boot Procedure

Normally, the user will let the boot procedure take its course and system operation will begin automatically. As part of the normal boot, UNIX checks its file system with *fsck*, and only completes the boot if the file system is intact. UNIX also checks the the status of all system hardware and, if everything looks good, comes up in standard multi-user mode.

After the Self Test is complete the monitor immediately attempts to boot from a default device (it tries a Xylogics SMD Disk Controller, then a SCSI Disk Controller, and finally tries to boot over the network):

```
Auto-boot in progress

Boot: disk(0,0,0)vmunix
Load: disk(0,0,0)boot
Boot: disk(0,0,0)vmunix
Size: 215040+24576+30916 bytes
Sun UNIX 4.2, etc...
```

disk is the device name of the 'best' local or network disk the monitor could find. The file called `vmunix` is booted from it. This file does not have to contain a UNIX kernel; it can contain any program you like, as long as the disk is in 4.2BSD UNIX file system format. It is also possible to set up the disk to boot a small program which need not be in a UNIX file system. This discussion assumes that the disk is set up for UNIX.

Booting From Specific Devices

The Sun Workstation can be booted from:

- Any logical partition of a local disk.
- Any publicly available network disk partition on your Ethernet.
- The first file of a local tape drive.

As mentioned above, the monitor automatically attempts to boot `vmunix` from a default disk. If you want to boot a different program, or from a different device, you must stop the automatic boot process by aborting. The specific abort sequence depends on your keyboard type. For a Sun-1 Model 100U or 150U, the sequence is either 'SET-UP-A' (hold down the 'SET-UP' key while typing the 'A' key) or 'ERASE-EOF-A'. For a Sun-2 keyboard, the abort sequence is 'L1-A'. For a standard terminal keyboard, the BREAK key generates an abort. When you abort, the monitor displays the address where it aborted and a **>** prompt, and waits for you to type a command.

The monitor's boot command looks like:

```
> b device(parameters)pathname args
```

where *device* is the type of hardware to boot from, *parameters* specify the address or partitioning of the device, *pathname* is the name of the actual file (in a UNIX file system on that device) to boot into memory, and *args* are optional arguments to the program.

To determine which devices your monitor PROM is able to boot from, you can use the command:

```
> b ?
```

The devices are shown in order from 'best' to 'worst', as used by the automatic boot procedure to select a boot device.

To boot from the monitor command level on the default device (the first device the monitor PROM can find to boot from), type:

```
> b
```

This is the normal command to give when you have interrupted automatic reboot and want to proceed to boot from `/vmunix`.

If you want to come up in single-user mode, type:

```
> b -s
```

IF you are up and running single-user, you can bring the system up to multi-user by typing the 'CTRL' key and the 'd' key simultaneously:

```
# ^D
```

Different ways of booting UNIX are used in various circumstances. Here are some of the ways; they are explained in the sections below.

- When booting multi-user fails, booting single-user will sometimes work. It may then be possible to fix the system from single-user mode so it will work again in multi-user mode. For instance, occasionally the `/etc/passwd`

file will become corrupted and no one can log into the multi-user system. If you boot single-user, you can fix the `/etc/passwd` file and re-boot multi-user. Remember, to go from single-user to multi-user, type **^D**.

- You may want to boot from a non-default boot device, particularly if your default device has been corrupted. Currently a system can be explicitly booted from disk, tape or network devices.
- From time to time it is necessary to boot a program from a tape or network because the system cannot access its own disk file system. For instance, the program `diag`(8) can be booted off tape so that the user can fix or re-format a bad disk.

Booting From Disk

For disk drives, the format of the boot command is:

> **b** *controller(address,drive,partition)pathname args*

controller	names the disk controller which runs the specific disk: **ip** for the Interphase 2180 disk controller, **xy** for the Xylogics 450 controller, or **sd** for a SCSI disk controller.
address	can either be a small number, indicating the *n*th standard controller board, or is the physical address of the controller on the Multibus.
drive	is the unit number of the disk on that specific controller.
partition	is a number corresponding to the logical partition on the disk where the file specified by *pathname* can be found. Zero corresponds to partition 'a', 1 to 'b', etc.
pathname	is the name of the file to boot.
args	are optional arguments.

Booting From Network Disk

To boot the system from network disk, use a command like:

> **b** *controller* **(***address, hostnumber, partition* **)** *pathname args*

controller	is the device abbreviation for your Ethernet Controller: **ec** for a 3COM Ethernet Controller, or **ie** for a Sun-2 Ethernet Controller.
address	can either be a small number, indicating the *n*th standard controller board, or is the physical address of the controller on the Multibus.
hostnumber	is an arbitrary number (between 1 and 255) assigned to each machine on a local network to uniquely identify the machine. To find the host number of an existing node, check the node's `/etc/hosts` database. The entries in the file look something like:

```
192.9.1.1      winkin
192.9.1.2      blinkin
192.9.1.3      nod
192.9.1.24     henry
```

The last component of the complete internet address is the host number. Henry's host number here is 24. Note that you must supply the hostnumber to the monitor in hexadecimal; if the numbers in the `/etc/hosts` database are in decimal, you will have to convert.

Using zero as *hostnumber* is valid here, and means 'whichever host is my net disk server.'

partition is the desired public partition number on the specified server. The correspondence between this number and a real disk partition is defined in `/etc/nd.local` on the server machine.

pathname is the name of the file to boot.

args are optional arguments.

Booting From Tape

The Sun Workstation can be booted from 1/2-inch nine-track magnetic tape, from a 1/4-inch cartridge tape controlled by a Sun 1/4-inch tape controller, or from a 1/4-inch tape controlled by a SCSI tape controller, using the following command:

> **b** *tape* (*controller, unit, filenum*)

tape is the device abbreviation for your tape controller: **mt** for 1/2-inch tape with a Tapemaster controller, **xt** for 1/2-inch tape with a Xylogics controller, **ar** for a Sun 1/4-inch tape controller, or **st** for a SCSI tape controller.

controller is a small number indicating the *n*th standard magnetic tape controller in the system, or is the Multibus address of the controller.

unit specifies which tape drive on the controller is to be used.

filenum specifies which file of the tape is to be booted. By convention, boot commands number the first file on the tape file #0, the second #1, and so on.

The monitor ignores the supplied value of *filenum* and can only boot the first file on a tape. To boot a file further down the tape, use the monitor to boot the 'boot' program. Sun-supplied UNIX distribution tapes always have the 'boot' program on the first file of the tape.

Booting Files From The Default Device

To boot any file from the default device, enter:

> **b** *pathname args*

In this manner you boot an alternate version of the kernel by supplying its name at *pathname* in the example above. This is also useful for booting standalone utility programs, like `diag`, after your disk or network disk is set up.

Shutdown Procedures

To shut down UNIX in any non-emergency situation, the super-user should execute `/etc/shutdown`(8). `shutdown` provides an automated shutdown procedure which notifies users that a system halt is pending. You may optionally specify a time and a message to be broadcast to all current users. In addition, you may optionally ask that a `halt`(8) or a `reboot`(8) be executed automatically after `shutdown` is complete. See the manual page entry for a description of all flags and options.

`shutdown` inhibits logins and, of course, executes `sync`(8) to ensure that all information is written to disk. At the time designated, the system is brought down to single-user mode.

When you need to bring down the system quickly and unexpectedly, you can execute `/etc/halt`. `/etc/halt` just writes out information to the disks and halts the processor — no warning, no delay, no mess. It takes you back to the monitor, out of UNIX. Using `halt` instead of `shutdown`, when an immediate system halt is not necessary, can be very disturbing to users.

When a UNIX system is running and rebooting is desired, `shutdown` is normally used. However, if you are running single-user, `/etc/reboot` can be used. `/etc/reboot` executes `sync` to write out information to disk, and initiates a multi-user reboot. Disk checks are performed with `fsck` and if all is well the system comes up multi-user.

Both `/etc/fastboot` and `/etc/fasthalt` are shell scripts which reboot and halt UNIX without checking the file systems. These should be used with care and understanding because they do not check file system consistency through `fsck`.

Power Loss

If UNIX has not attempted to write to the file system for about 30 seconds prior to power loss, chances are that nothing abnormal will happen when you try to re-power the system — no guarantees on this, since power loss may always cause problems.

If you lose power before UNIX has had a chance to complete all outstanding disk writes, or if you halt the system without using `/etc/halt`, `/etc/shutdown`, or `sync`, UNIX may complain when it checks its file system with the `fsck` program.

You should always check file system consistency with `fsck` when re-booting. It will attempt to make corrective changes where it detects problems. You should let it do so, by answering 'yes' to questions it asks you. During `fsck`, some unreferenced files may be placed in the `lost+found` directory. See the manual entry for `fsck` for more information about how this works and how to retrieve files placed there. If you do not allow `fsck` to make its changes, your file system will be in an unreliable state when you finally boot UNIX. This can have disastrous consequences!

Messages from the Monitor and the Boot Program

Abort at *aaaaaa*

The monitor has aborted execution of the current program because you entered the 'abort sequence' (upper left key held while pressing 'A') from the Sun keyboard, or pressed BREAK on a serial console. *aaaaaa* is the address of the next instruction. You can continue the program from there by entering the c command.

Address Error, addr: *xxxxxx* at *aaaaaa*

The current program has stopped because it made an invalid memory access. *xxxxxx* is the (invalid) address; *aaaaaa* is an address near the instruction which failed (typically two to ten bytes beyond). There is no general way to recover from this error, except to debug the program.

ar: cartridge is write protected

The current program is trying to write on an Archive tape cartridge, but the 'Safe' switch at the top left corner of the cartridge is set to prevent writing on the tape.

ar: *xxxx* error

The monitor or boot program is trying to boot from an Archive tape, and encountered an unexpected error. The status bytes *xxxx* can be decoded by looking under 'Read Status Command' in the *Archive Product Manual.* This error could be caused by incorrect cables, a bad tape, or other problems.

ar: drive not responding

The monitor is trying to boot from an Archive tape, but can get no response from the tape drive. This can occur if your system contains an Archive controller board but no tape drive, or if the tape drive's cable is loose or disconnected, or if the tape drive's power is not on.

ar: invalid state *xx*

This message indicates that the standalone I/O system has a bug in its Archive driver.

ar: no cartridge in drive

The monitor or boot program is trying to boot from an Archive tape, but there is no cartridge in the tape drive.

ar: no drive

The monitor or boot program is trying to boot from an Archive tape, but the specified drive does not exist. Typical Archive configurations include only drive 0.

ar: RDST gave Exception, retrying

The current program is trying to use the Archive tape drive, and encountered an error. The error is probably caused by hardware. Check the cable(s) that connect the tape drive to the system.

ar: triggered at idle *xx*

This message indicates that the standalone I/O system has a bug in its Archive driver.

Auto-boot in progress...
The monitor has finished its power-on sequence and is looking for a good device to boot the UNIX system from.

Bad device
The current program (possibly the boot program) has tried to open a file without a device name (for example, `xy()`). This could mean that the boot command you typed had no device name.

Bad format
The boot program is trying to boot from a file which is not in a standard UNIX `a.out` (5) format. The boot program can only boot files which are in this format, which is generated by the `ld` (1) command.

bn negative

bn ovf `dd`

bn void `dd`
A standalone program (such as the boot program) is trying to read a file from disk or net disk, and the block number it is trying to read is invalid.

Boot:
The boot program is waiting for you to specify a device and file name to boot from. The boot program accepts the same commands that the monitor would, without the initial 'b'. See the section *Booting From Specific Devices* above.

Boot: *dev(ctlr,unit,part)name options*
The monitor or boot program is preparing to boot the specified file from the specified device. Either you typed a boot command, or this is an auto-boot after power-on. *dev* is the device type; *ctlr, unit*, and *part* are the controller, unit-within-controller, and disk partition number. *Name* is the name of the file to boot from, if any; *options* are arguments for the booted program, such as '-s'. If you enter a boot command to the monitor, this message will be printed twice; once by the monitor and once by the boot program.

boot failed
The boot program has tried to boot the device and/or file you specified, but could not. A preceding message should give more details about why.

Boot syntax: b [!][dev(ctlr,unit,part)] name [options]

boot syntax: dev(ctlr,unit,part)name
You have entered an invalid boot command. This message describes the general format of the boot command to remind you. The first form is used by the monitor; the second (without the 'b') is used by the boot program. Don't type the brackets; they indicate optional parts of the command.

Bus Error, addr: *xxxxxx* at *aaaaaa*
The current program has stopped because it tried to make an invalid memory access. The reason for the error is shown before this message. The memory location being accessed was *xxxxxx*, and the instruction which made the access is near location *aaaaaa*. There is no way to recover from this error,

in general, except to debug the program.

Can't write files yet...Sorry
: The current program is trying to write to a disk or network disk file thru the standalone I/O system. Writing on files (as opposed to writing on devices) is not supported when running standalone (that is, before booting the UNIX kernel).

Corrupt label

Corrupt label on head *h*
: The monitor or boot program is trying to boot from a disk. The first sector of the disk appears to be a label (as it ought to be), but the checksum on the label is wrong. Try again a few times; if the problem recurs, you should probably relabel your disk. See sections *Using the Diag Utility* and *Labeling the Disk* in the chapter, *Installing UNIX for the First Time*. Before trying to relabel your disk, make sure that you know what ought to be in the label — writing the wrong label on the disk is highly likely to cause destruction of some or all files on the disk.

count=*ddd*?
: A standalone program is trying to write to a device and has specified a block size which is not a multiple of 512. The write proceeds anyway, but may cause incorrect results.

Damage found, *damage...*
: As part of the power-on self test procedure, the monitor has found damage in one or more parts of the system. This message should be reported to your local service representative or Sun Microsystems Field Service. *Damage* is a list of subsystem names, such as 'memory' or 'timer'.

Exception *ee* at *aaaaaa*
: The current program has stopped because it got an interrupt. The interrupt could have been caused either by hardware or software. *ee* is the hexadecimal address of the interrupt vector used; you can look it up on a Motorola 68010 or 68000 reference card or CPU manual to see what kind of interrupt has occurred. *aaaaaa* is the address of the instruction where the interrupt occurred. If *ee* is `2c`, the partition you are trying to boot from is probably missing its boot track.

Extra chars in command
: Your previous 'u' command had extra, unrecognized characters on the end.

FC*n* space
: The address space being accessed by the monitor's memory reference commands is defined by Function Code number *n*. See the Motorola 68010 CPU manual for more information. This message is printed by the 's' command.

For phys part *p*, No label found.
: The boot program is trying to boot from a non zero 'physical partition' on a disk, and can't find a label. Physical partitions are used for disk drives part of which are fixed and part of which are removable.

—> Give the above information to your service-person.
The monitor has found a hardware problem while executing its power-on self test procedure. The preceding messages describe the error in more detail. You should report the problem to your local service staff, or to Sun Microsystems Field Service.

Giving up...
See "Waiting for disk to spin up...". The monitor has given up on waiting for the disk to become ready.

ie: cannot initialize
The monitor or boot program is trying to boot from a Sun-2 Ethernet controller, and something serious has gone wrong with the board. Call your local Sun Microsystems Field Serviceperson.

ip: error *xx*
The monitor or boot program is trying to boot from an Interphase disk controller, and encountered an unexpected error. The error number *xx* can be decoded by looking in appendix B of the Interphase *SMD 2180 Storage Module Controller/Formatter User's Guide*. This problem is usually caused by loose or unplugged disk drive cables.

ID PROM INVALID
The monitor cannot find a valid ID PROM on the CPU board. The ID PROM contains the machine's serial number and other information specific to your system. If you have recently changed CPU boards, it is possible that you installed the ID PROM incorrectly. You should attempt to locate the correct ID PROM and install it in your CPU board.

Invalid Page Bus Error ...
See "Bus Error...". The attempted access was invalid because the virtual page containing the addressed data has been designated as invalid. It usually means that your program is using the wrong address.

Invalid selection
Your last 'u' command was not correct.

Keyboard error detected
The microprocessor on the keyboard has reported an error. This probably means that your keyboard hardware is broken and should be replaced.

Load: *dev*(*ctlr*,*unit*,*part*)boot
The monitor has loaded in the 'mini' boot program from a disk drive or network disk. The mini boot is now reading in the 'real' boot program from the disk. The 'real' boot program will then read in the program you requested.

Lower Byte Parity Bus Error ...
See "Bus Error..." and "Parity...". The preceding access was to a word in memory with a parity error in its lower byte.

Misplaced label on head *n*
The monitor or boot program is trying to boot from a disk. It has found a label which seems to identify itself as belonging to a different read/write head from the one where the label is written. See "Corrupt label..." above.

mt: controller does not initialize
The monitor is trying to boot from nine-track tape, and could not get the tape controller to complete its initialization sequence. This might indicate a possible defect in the controller, or incorrect configuration of the controller board.

mt: error 0x*xx*
The monitor is trying to boot from nine-track tape, and encountered an unexpected error. The error number *xx* can be decoded by looking in *Appendix C* of the *Tapemaster Product Specification*, which is supplied with your tape drive.

mt: unit not ready
The monitor is trying to boot from nine-track tape, but the tape drive is not ready. Check to see that the drive is on-line.

nd: no file server, giving up.
The monitor or boot program is trying to boot from a network disk server over the Ethernet. It has been retrying for a long time and there is no response from the server. Check the Ethernet address in the boot command; if it is zero, make sure your machine's Ethernet address is recorded in the server's `/etc/nd.local` file. If that's OK, check your Ethernet cable connection, see whether the server is running correctly, and/or see whether other machines on the network can communicate.

No controller at mbio *xxxx*
The monitor is trying to boot, but it can't find a device controller where you asked it to look. You should try another boot command, or make sure that your controller board is plugged in and has all its jumpers and switches set properly.

No default boot devices
The monitor is trying to boot but it can't find a disk or Ethernet interface to boot from. To boot from a tape, you must specify the device name explicitly, as in '`b ar()`'.

No label found -- attempting boot anyway.
The monitor or boot program is trying to boot from a disk, and can't find a valid label on the disk. This is best fixed by booting a copy of `diag`(8S) from a different device (for example, network disk or tape) and using the 'verify label' and 'label' commands. See the warning under 'Corrupt label' above. This error might also be caused by missing or bad disk cables.

No more file slots
The current program is using the standalone I/O library and has opened too many devices or files.

not a directory
The current program (possibly the boot program) has tried to open a disk or network disk file with a pathname, but one of the names in the path is not a directory.

name not found
The boot program has searched for the requested file, but cannot find it. You can retry your boot command, using * instead of *name*, to get a list of the names that exist in that directory.

null path
The current program (possibly the boot program) has tried to open a file whose name is empty.

PageMap *aaaaaa* [*ss*]: *xxxxxxxx*?
The monitor is displaying or modifying a page map entry because you entered a 'p' command. *aaaaaa* is the virtual memory address whose map entry is being examined. *ss* is the segment map entry which is being used to map this page map entry and page. *xxxxxxxx* is the page map entry itself. You can enter a space and type 'RETURN' to get back to command mode.

Parity Bus Error ...
See "Bus Error...". The attempted access was probably valid, but was canceled because the preceding access was to memory with bad parity. (Parity errors are reported on the memory cycle **after** the failing cycle.) If neither 'Upper Byte' nor 'Lower Byte' is reported, the parity on both bytes was invalid. The access address printed in the Bus Error message is probably not relevant to the parity error. There is no general way to recover from this error; a good starting point, though, is to boot `parscan` (8S), which will search all of memory for parity errors.

Please clear keyboard to begin
The monitor is trying to listen for your typing on the keyboard, but cannot tell which shift keys are down until you release all the locking keys (Caps Lock and Shift Lock). Once it has seen all the keys released, it can then track the movements of the keys and typing will work.

Please start it, if necessary, -OR- press any key to quit.
See "Waiting for disk to spin up...".

Possible boot devices:
You have asked for a list of boot devices with the '`b ?`' command.

Protection Bus Error ...
See "Bus Error...". The attempted access was invalid because your program is not permitted to access the addressed data in this way; for example, writing to that page is disallowed.

ROM Rev *x*, *mm* memory installed
The monitor is identifying its revision level and the system configuration as part of the power-on sequence. *x* is a letter or phrase indicating which particular version of the monitor is installed; *mm* shows how much memory was found during system configuration at power-on. If an even number of megabytes is installed, *mm* is displayed as *nn*MB; otherwise as *nnnn*KB.

Retensing...
The monitor is attempting to boot from an Archive tape. Its first attempt failed, so it is retensing the tape (winding all the tape from one reel to the

other), which makes it much more likely to succeed.

Seek not from beginning of file
The current program is using the standalone I/O library and has tried to do an unsupported seek operation.

SegMap *aaaaaa*: *xx*?
The monitor is examining or changing the segment map in response to your recent 'm' command. You can enter a space and type 'RETURN' to get back to command mode.

Self Test completed successfully
The monitor has completed its power-on self test without finding any hardware problems.

Self Test found a problem in *something*
The monitor has completed its power-on self test and found a problem in some subsystem. *Something* describes the general location of the error. Further messages give more details; see "Wrote ..." and "Damage found...".

Serial #*some_number*, Ethernet address *n:n:n:n:n:n*
The monitor is identifying your machine's serial number and hardware Ethernet address as part of the power-on sequence. The hardware Ethernet address is taken from the ID PROM on the Sun-2 CPU board, and is given as a 6-byte hexadecimal value with a colon between each byte. A typical Ethernet address might be `8:0:20:1:1:A3`.

Short read
The boot program is trying to boot a program from disk or net disk. It has located the program, but encountered an error while reading it into memory.

Size: *text* +*data* +*bss* bytes
The boot program is loading in the program you requested. *Text*, *data*, and *bss* are the sizes of the three sections of the program; they are printed as each is read into memory. After finishing display of this message, the boot program begins execution of your program; further messages can come from it instead of from the boot program or monitor.

Sun Workstation, Model Sun-1/100U or Sun-1/150U, *keyb* keyboard
The Model 100U or 150U workstation has just been powered on, or you entered a 'kb' command, and the monitor is identifying its configuration. *Keyb* is either VT100 or Two-tone, depending which keyboard your monitor ROMs support.

Sun Workstation, Model Sun-2/120 or Sun-2/170, Sun-2 keyboard
The Model 120 or 170 workstation has just been powered on, or you entered a 'kb' command, and the monitor is identifying its configuration.

Timeout Bus Error ...
See "Bus Error...". The attempted access was invalid because no device responded at the addressed location. This most often happens for Multibus references. The program was probably trying to access a device or section of memory which does not exist, or which has gotten into a hung state. If

this occurs in response to a boot command, the device you are trying to boot from is not installed in your system.

tm: error *nn* during config of ctlr *cc*

tm hard err *nn*

tm: no response from ctlr *cc*
A standalone program (possibly the boot program) is trying to use the Tapemaster nine-track tape drive, and has encountered an error. This could be caused by a bad or missing tape, loose or misplugged cables, incorrect jumpers on the Tapemaster controller board, or hardware errors. *Nn* can be decoded by looking in the Tapemaster *Product Specification.*

Unknown device
The current program (possibly the boot program) has tried to use a device which is unknown to the standalone I/O system.

Upper Byte Parity Bus Error ...
See "Bus Error..." and "Parity...". The preceding access was to a word in memory with a parity error in its upper byte.

u*i* i, u*o*o, ua*abaud*, ub*bbaud*, uu*aaaaaa*, u*echo*
The monitor is describing its console and serial port configuration in response to a 'u' command. *i* is the input device (`k` for keyboard, or `a` or `b` for a serial port); *o* is the output device (`s` for screen, or `a` or `b`); *abaud* and *bbaud* are the baud rates on the serial ports; *aaaaaa* is the address of the Zilog 8530 chip which implements the serial ports, and *echo* is `e` if input echoing is enabled (full duplex) or `ne` if disabled (half duplex).

Using RS232 A input.
The monitor did not find the Sun keyboard, so it is taking input from one of the serial ports on the back of the Workstation, marked 'RS232 A'. If this is unexpected, make sure that the keyboard is plugged into the correct socket on the workstation. The keyboard must be plugged in before system power is turned on. If you connect a Sun keyboard after this message appears, you can let the monitor know about the keyboard by entering the Abort sequence (hold down the upper left key on the Sun keyboard, and press 'A'). The monitor will switch to using the Sun keyboard since that's where the Abort was typed. Then type `c` to continue whatever program was running when you aborted.

If you don't want to use a Sun keyboard, connect a normal ASCII terminal to the 'RS232 A' connector on the back panel. Configure the terminal for 9600 baud, no parity, one stop bit. Things that you type on the terminal will be displayed on the Sun video screen, if you have one, or on the terminal's screen.

Waiting for disk to spin up...
The monitor is trying to boot from a disk. The disk is not ready, so the monitor is waiting in the hope that the disk is just starting to spin and will become ready soon. If you get this message when the power has been on for a while, your disk cables are probably loose or misconnected.

Watchdog reset!
The current program has stopped executing with a 'double bus fault.' This is explained in detail in the Motorola 68010 manual; the two most common causes are that low memory (interrupt vectors) has been overwritten, or the system stack pointer is pointing to an invalid address. There is a serious problem, possibly in the kernel you are running, more likely it is in the hardware.

What?
You typed a command that the monitor does not recognize. Try again.

Wrote *wdata* at address *addr*, but read *rdata*
The monitor has completed its power-on self test and found a problem in some subsystem. The preceding "Self Test found a problem..." message describes which part of the system was in error. This message gives more details about the error. *wdata* is the data that was written into part of the system, or which was expected to be there if the system was functioning normally. *addr* is the address where the data was read and/or written. For memory errors, this is a physical memory address; for other errors, the interpretation of this field depends on what subsystem was being tested. *rdata* is the data that was read back from *addr* and was found to be invalid because it was not the same as *wdata*. This information should be written down and reported to your local Field Service organization, or to Sun Microsystems Field Service. See the section *Non-Critical Errors From Self Test* above.

xt: error *nn* during config of ctlr *cc*

xt hard err *nn*

xt: no response from ctlr *cc*
A standalone program (possibly the boot program) is trying to use the Tapemaster nine-track tape drive, and has encountered an error. This could be caused by a bad or missing tape, loose or misplugged cables, incorrect jumpers on the Tapemaster controller board, or hardware errors. *Nn* can be decoded by looking in the Xylogics Product Specification.

xy: error *nn* cmd *xx*

xy: error *nn* bno *bbbbb*

xy: init error *xx*
The monitor is trying to boot from the Xylogics disk and has encountered an error. The command being executed at the time is defined by the hexadecimal value *xx* (if present); the block number is *bbbbb* (if present), and the particular error is encoded as *nn*. The error and command can be decoded by looking in the Xylogics manual.

xy: no bad block info
The boot program is trying to read from the Xylogics disk, but can't find the information about bad blocks on the disk. It continues, but if the program attempts to read any bad blocks (which have been remapped to elsewhere on the disk), the attempt will fail.

zero length directory
: A standalone program (possibly the boot program) is trying to read a file from disk, but one of the directories in the path name has no files in it. The file system should be checked and fixed by using `fsck` (8).

6.3. When The System Crashes

Sooner or later, every system will crash or become hung in such a way that it no longer responds to commands. This section discusses some of the ways a system administrator can respond to system crashes and hung systems.

When the system crashes it prints out a short message telling why it crashed, attempts to preserve a core image, and invokes an automatic reboot procedure. If the reboot finds no unexpected inconsistency in the file systems due to software or hardware failure, it will resume multi-user operations. Below we discuss the error messages generated by a crash. In addition we discuss how the system core dump is saved, how to get rid of unwanted core dumps that can accumulate on your system, how to force a core dump, what steps can be taken to analyze a core dump, what to do if the automatic reboot fails, and when to call for help.

In general, a bad program should never crash the system. If it does, there is probably a bug in the kernel. Sometimes, however, a user program may crash and 'hang' something in the system — a user process, a user's window, or even the whole system — even though UNIX continues to run. Some aspects of a hung system and how to overcome them are discussed below. As the section *User Program Crashes* explains below, you are sometimes forced to reboot after a crashed program, even though UNIX has not crashed.

Crash Error Messages

When the system crashes, it prints out a message like:

```
panic: what I think went wrong
```

Less frequently you might see the message `Watchdog reset!` in place of the `panic`. A list of the most likely messages, along with a short explanation, can be found on the manual page `crash(8)`. Where there are repeated crashes it is extremely important to keep an exact copy of each message — including punctuation and upper or lower case lettering. These will be helpful to a doctor trying to heal a sick system.

Two files automatically store messages about crashes: `/usr/adm/messages`, keeps a record of system messages, and `/usr/adm/shutdownlog` tells how the system was shut down each time.

If you have not been able to obtain a copy of a crash message from the console, look in these files for a history of system behavior. They might also be helpful for determining patterns of problems over time.

System Core Dumps

As currently distributed, the system will not attempt to write a core dump. However, you can enable core dumps by editing the `/etc/rc.local` file on your machine. The lines to do the core dump are there, but are commented out for distribution. To enable core dumps look for the following lines in `/etc/rc.local`:

```
#
# Default is to not do a savecore
# Diskless clients should dump to /usr2 since
# this will generally have more free space than /usr
#
#mkdir /usr2/crash/`hostname`
#/usr/etc/savecore /usr2/crash/`hostname`     >/dev/console
```

And remove the '#' in front of the two command lines:

```
mkdir /usr2/crash/`hostname`
/usr/etc/savecore /usr2/crash/`hostname`     >/dev/console
```

Note that diskful machines will dump to `/usr/crash`, while `nd` servers and diskless clients will dump to `/usr/crash/`*hostname*, where *hostname* is the client machine name, as shown in this example. If you make the above change, the following information in this section will be true.

When the system crashes it writes, or attempts to write, an image of memory into the primary paging partition on disk (or on the network disk). After the system is rebooted, the program `savecore (8)` runs and preserves a copy of this core image in the directory `/usr/crash/`*hostname* (or `/usr/crash`). In most cases the system will reboot automatically after a crash, cleaning up any problems and allowing work to go on as before. Under these circumstances there will be little, if any, interest in preserving the core dump.

If core dumps are allowed to accumulate unchecked in `/usr/crash/`*hostname* (or `/usr/crash`), they will eventually fill up the file system. There is a convenient way to avoid this problem. The `savecore` program reads from a file in `/usr/crash/`*hostname* (or `/usr/crash`) named `minfree`, which contains a single number (in ASCII). If the file system contains fewer free kilobytes than the number in `minfree`, then the core dump will not be saved. If you do use `minfree` to prevent saving core dumps, remember to lower its value, or remove the file entirely, during those times when you want to save the core image for debugging.

Sometimes your machine will be hung without crashing. A core dump can usually be forced in this situation. First abort to the PROM monitor by typing the appropriate abort sequence for your keyboard. Then type:

```
> g 0
```

The dump will be placed in the `/usr/crash/`*hostname* (or `/usr/crash`) directory according to the `savecore` procedure explained above. However, you cannot force a core dump in this manner unless you have edited your `/etc/rc.local` file as explained above.

Sometimes the system can get so badly hung that a dump cannot be forced. This is especially true on diskless machines, since a lot of the network machinery has to work in order to do the dump.

Analyzing System Core Dumps

It is not recommended that novices attempt to debug UNIX from the the core dump. Sophisticated users can `analyze(8)` the dump, or run `adb(1)` with the **`-k`** flag to poke around in it. See *Using ADB to Debug the UNIX Kernel* in the *System Internals Manual* for more details.

User Program Crashes

User program crashes, as distinguished from UNIX system crashes, are almost always due to programmer error — a bug in the program. The most common messages from a program crash are: `Segmentation Fault`, `User Bus Error`, `Floating Point Exception`. These often indicate an illegal pointer in the program, perhaps the result of using a value where a pointer to a value was expected. System error messages are documented in the introduction to *System Calls*, Section 2 of the *System Interface Manual*. A crashing program will usually dump core in its current directory if it has write permission. If it does, you will see the message `Core Dumped`.

Sometimes, the system will appear hung or dead; that is to say it will not respond to keyboard input. Before making the, rather drastic, assumption that your program has crashed, check the items below to make sure that there is not some other simpler reason for the system's non-responsiveness. (When we use the circumflex ^ below it signifies holding down the CTRL key while typing the key shown. For example, **^Q** means to hold down the CTRL key and simultaneously strike **Q**.)

1) Type **^Q** (CTRL Q) in case **^S** was accidentally hit, freezing the screen.
2) The `tty` mode may be fouled up. Try typing the linefeed character, **^J** (CTRL J), instead of the RETURN key, to force a linefeed. If the system responds, type **^J /usr/ucb/reset ^J** to reset the `tty` modes.
3) If you are running *Suntools*, make sure the mouse cursor resides in the window where you are trying to type commands
4) Type **^** (CTRL backslash). This should force a 'quit' in the running program, and probably the writing of a 'core' file.
5) Type **^C** (CTRL C) to interrupt the program that may be running.
6) If possible try logging into the same CPU from another terminal, or `rlogin` from another system on the network. Type **`ps -ax`** and look for the hung process. If you can identify it, try to kill it. You will have to be superuser or be logged in as the same user running the process. Type **`kill`** *<pid number>*. If that does not work try **`kill -9`** *<pid>*. (A quick way to see if a **`kill`** has worked is to repeat it. If the response is `no such process`, it was killed.)
7) If all of the above fail, the system has probably gone to another world. Abort and reboot.
8) If even that fails, call Tech Support for help.

When To Call For Help

Before calling for help, make sure you have accurately copied down crash messages from the console, or taken them from the two `/usr/adm` files mentioned above.

If the automatic reboot fails with a message such as:

```
reboot failed: help
```

do not attempt to proceed. Call for Tech Support help.

If you are having frequent crashes, gather all the information you can about them, and have it ready when you call for help.

6.4. Kernel Configuration

Every UNIX site should configure the kernel when it is installed. You will find a detailed explanation and notes about the configuration process in the manual *Installing* UNIX *On The Sun Workstation.* This section supplements that discussion with information less frequently needed. It is not intended to be used without expert knowledge of the procedures discussed in that installation manual.

In addition, various sections in this *System Administration For The Sun Workstation* manual give brief, walkthrough explanations of common types of kernel reconfiguration. If you need to supplement those with broader context, you should look first at the kernel configuration section in *Installing* UNIX *On The Sun Workstation*, since the sub-sections that follow here are mostly rather esoteric and specific.

Notes to Step 3 of Kernel Configuration

This section contains four additional notes about the procedures in Step 3 of kernel configuration, as they are given in the manual *Installing* UNIX *On The Sun Workstation.* Briefly, the notes cover:

1) System configuration on systems without source code
2) Adding new device drivers
3) Sharing object modules
4) Building profiled kernels.

Note 1: Configuring Systems Without Source

Object-only releases have binaries for standard system modules in the directory `/sys/OBJ`. Using these binaries you can create new configurations and add new device drivers to the kernel. The following lines from the GENERIC config file must be in every config file for object-only distributions:

```
machine sun
cpu "SUN2"
options "INET"

pseudo-device   inet
pseudo-device   ether
pseudo-device   loop
controller  mb0 at nexus ?
```

If you include these lines you can make any changes you wish to the configuration file, provided you do not configure in more devices of a particular type than are allowed by the distributed object code in `/sys/OBJ`. Attempting to do so will not be detected and may cause the kernel to appear to work but have only occasional failures. Double check the `.h` files in `/sys/OBJ` if you change the number of devices configured for any standard drivers.

Note 2: Adding New Device Drivers

New device drivers require entries in the files `/sys/sun/conf.c`, `/sys/conf/files.sun`, and possibly `/sys/sun/swapgeneric.c` and `sys/conf/devices.sun`. New devices also require one or more new special files to be added to the `/dev` directory. See the *Device Driver Tutorial* in the Sun *System Internals Manual.*

Note 3: Sharing Object Modules

This is only effective if you have a source distribution.

If you have many kernels which are all built on a single machine there are at least two approaches to saving time in building kernel images. The best way is to have a single kernel image which is run on all machines. This is attractive since it minimizes disk space used and time required to rebuild kernels after making changes. However, it is often the case that one or more systems will require a separately configured kernel image. This may be due to limited memory (building a kernel with many unused device drivers wastes core), or to configuration requirements (one machine may be a development machine where accounting is not needed, while another is a production machine where it is), etc. In these cases it is possible for kernels to share relocatable object modules which are not configuration dependent; most of the modules in the directory `/sys/sys` are of this sort.

To share object modules, first build a GENERIC kernel. Then, for each kernel, configure as before, but before recompiling and linking, type `make links` (specifically, after the `make depend` command and before the `make` that makes your kernel, as explained in *Installing UNIX On The Sun Workstation*). This will cause the kernel source to be searched for source modules which are safe to share between kernels and generate symbolic links in the current directory to the appropriate object modules in the directory `../GENERIC`. A shell script, `makelinks` is generated and executed with this request and may be checked for correctness. The file `/sys/conf/defines` contains a list of symbols which we believe are safe to ignore when checking the source code for modules which may be shared. Note that this list includes the definitions used to conditionally compile in the virtual memory tracing facilities, and the trace point support used only rarely. It may be necessary to modify this list to reflect local needs. Also, as described previously, interdependencies which are not directly visible in the source code are not caught. Thus if you place per-system dependencies in an include file, they will not be recognized.

Note 4: Building Profiled Kernels

This is only effective if you have a source distribution.

It is simple to configure a kernel which will automatically collect profiling information as it operates. The profiling data may be collected with `kgmon` (8) and processed with `gprof` (1) to obtain information regarding the kernel's operation. Profiled kernels maintain histograms of the program counter as well as the number of invocations of each routine. The `gprof` (1) command will generate a dynamic call graph of the executing kernel and propagate time spent in each routine along the arcs of the call graph.

To configure a profiled kernel, use the `-p` option with the `config` program. A profiled kernel is about 5-10% larger in its text space due to the calls to count the subroutine invocations. When the kernel executes, the profiling data is stored in a buffer which is 1.2 times the size of the text space. The overhead for running a profiled kernel varies; under normal load we see anywhere from 5-25% of the kernel time spent in the profiling code.

Note that kernels configured for profiling should not be shared as described above unless all the other shared kernels are also to be profiled.

Rules for Defaulting System Devices

This section covers the rules the `config` program uses to define underspecified locations of system devices.

When the `config` program processes a `config` line which does not fully specify the location of the root file system, swap or paging area(s), device for system dumps, and device for argument list processing it applies a set of rules to define those values left unspecified. The following list of rules is used in defaulting system devices.

1) If a root device is not specified, the swap specification must indicate a 'generic' system is to be built.
2) If the root device does not specify a unit number, it defaults to unit 0.
3) If the root device does not include a partition specification, it defaults to the `a` partition.
4) If no swap area is specified, it defaults to the `b` partition of the root device.
5) If no device is specified for processing argument lists, the first swap partition is selected.
6) If no device is chosen for system dumps, the first swap partition is selected (see below to find out where dumps are placed within the partition).

The following table summarizes the default partitions selected when a device specification is incomplete, e.g. `xy0`.

Type	Partition
root	a
swap	b
args	b
dumps	b

In addition, if/when multiple swap partitions are specified, the system treats the first specified as a 'primary' swap area which is always used. The remaining partitions are then interleaved into the paging system at the time a `swapon` (2) system call is made. This is normally done at boot time with a call to `swapon` (8) from the `/etc/rc` file.

Finally, system dumps are automatically taken after a system crash, provided the device driver for the 'dumps' device supports this. The dump contains the contents of memory, but not the swap areas. Normally the dump device is a disk in which case the information is copied to a location near the back of the partition. The dump is placed in the back of the partition because the primary swap and dump device are commonly the same device and this allows the system to be rebooted without immediately overwriting the saved information. When a dump has occurred, the system variable `dumpsize` is set to a non-zero value indicating the size (in bytes) of the dump. The `savecore` (8) program then copies the information from the dump partition to a file in a 'crash' directory and also makes a copy of the system which was running at the time of the crash (usually `/vmunix`).

Configuration File Grammar

This section covers the grammar used by the `config` program to parse the input kernel configuration file.

The following grammar is a compressed form of the actual `yacc` (1) grammar used by the `config` program to parse configuration files. Terminal symbols are shown all in upper case, literals are emboldened; optional clauses are enclosed in brackets, `[` and `]`; zero or more instantiations are denoted with `*`.

```
Configuration ::=  [ Spec ; ]*

Spec ::= Config_spec
    | Device_spec
    | trace
    | /* lambda */

/* configuration specifications */

Config_spec ::=  machine ID
    | cpu ID
    | options Opt_list
    | ident ID
    | System_spec
    | timezone [ - ] NUMBER [ dst [ NUMBER ] ]
    | timezone [ - ] FPNUMBER [ dst [ NUMBER ] ]
    | maxusers NUMBER

/* system configuration specifications */

System_spec ::= config ID System_parameter [ System_parameter ]*

System_parameter ::=  swap_spec | root_spec | dump_spec | arg_spec

swap_spec ::=  swap [ on ] swap_dev [ and swap_dev ]*

swap_dev ::=  dev_spec [ size NUMBER ]

root_spec ::=  root [ on ] dev_spec

dump_spec ::=  dumps [ on ] dev_spec

arg_spec ::=  args [ on ] dev_spec

dev_spec ::=  dev_name | major_minor

major_minor ::=  major NUMBER minor NUMBER

dev_name ::=  ID [ NUMBER [ ID ] ]

/* option specifications */

Opt_list ::=  Option [ , Option ]*

Option ::=  ID [ = Opt_value ]

Opt_value ::=  ID | NUMBER

/* device specifications */

Device_spec ::= device Dev_name Dev_info Int_spec
    | disk Dev_name Dev_info
```

```
    | tape Dev_name Dev_info
    | controller Dev_name Dev_info [ Int_spec ]
    | pseudo-device Dev [ NUMBER ]

Dev_name ::=  Dev NUMBER

Dev ::=  uba | mb | ID

Dev_info ::=  Con_info [ Info ]*

Con_info ::=  at Dev NUMBER
    | at nexus NUMBER

Info ::=  csr [ bus_spec space_spec ] NUMBER
    | drive NUMBER
    | slave NUMBER
    | flags NUMBER

Bus_spec ::=  all | mb | vme

Space_spec ::=  virt
    | obmem
    | obio
    | busmem
    | busio

Int_spec ::=  priority NUMBER
    | priority NUMBER vector ID NUMBER [ ID NUMBER ] *
    | /* epsilon */
```

Lexical Conventions

The terminal symbols are loosely defined as:

`ID`
: One or more alphabetics, either upper or lower case, and underscore, _.

`NUMBER`
: Approximately the C language specification for an integer number. That is, a leading `0x` indicates a hexadecimal value, a leading `0` indicates an octal value, otherwise the number is expected to be a decimal value. Hexadecimal numbers may use either upper or lower case alphabetics.

`FPNUMBER`
: A floating point number without exponent. That is a number of the form `nnn.ddd`, where the fractional component is optional.

In special instances a question mark, `?`, can be substituted for a `NUMBER` token. This is used to effect wildcarding in device interconnection specifications.

Comments in configuration files are indicated by a # character at the beginning of the line; the remainder of the line is discarded.

A specification is interpreted as a continuation of the previous line if the first character of the line is tab.

Data Structure Sizing Rules

This section explains the rules for sizing kernel data structures, both those calculated at compile time and those calculated at boot time.

Certain kernel data structures are sized at compile time according to the maximum number of simultaneous users expected, while others are calculated at boot time based on the physical resources present, such as memory. This section lists both sets of rules and also includes some hints on changing built-in limitations on certain data structures.

Compile Time Rules

The file `/sys/conf/param.c` contains the definitions of almost all data structures sized at compile time. This file is copied into the directory of each configured kernel to allow configuration-dependent rules and values to be maintained. The rules implied by its contents are summarized below (here `MAXUSERS` refers to the value defined in the configuration file in the `maxusers` rule).

nproc
The maximum number of processes which may be running at any time. It is defined to be 10 + 16 * `MAXUSERS` and referred to in other calculations as `NPROC`.

ntext
The maximum number of active shared text segments. Defined as 24 + `MAXUSERS`.

ninode
The maximum number of files in the file system which may be active at any time. This includes files in use by users, as well as directory files being read or written by the system and files associated with bound sockets in the UNIX ipc domain. This is defined as (`NPROC` + 16 + `MAXUSERS`) + 32.

nfile
The number of 'file table' structures. One file table structure is used for each open, unshared, file descriptor. Multiple file descriptors may reference a single file table entry when they are created through a `dup` call, or as the result of a `fork`. This is defined to be

```
16 * (NPROC + 16 + MAXUSERS) / 10 + 32
```

ncallout
The number of 'callout' structures. One callout structure is used per internal system event handled with a timeout. Timeouts are used for terminal delays, watchdog routines in device drivers, protocol timeout processing, etc. This is defined as 16 + `NPROC`.

nclist
The number of 'c-list' structures. C-list structures are used in terminal i/o. This is defined as 100 + 16 * `MAXUSERS`.

nmbclusters
The maximum number of pages which may be allocated by the network. This is defined as 256 (a half megabyte of memory) in /sys/h/mbuf.h. In practice, the network rarely uses this much memory. It starts off by

allocating 64 kilobytes of memory, then requesting more as required. This value represents an upper bound.

nquota
The number of 'quota' structures allocated. Quota structures are present only when disc quotas are configured in the system. One quota structure is kept per user. This is defined to be (MAXUSERS * 9) / 7 + 3.

ndquot
The number of 'dquot' structures allocated. Dquot structures are present only when disc quotas are configured in the system. One dquot structure is required per user, per active file system quota. That is, when a user manipulates a file on a file system on which quotas are enabled, the information regarding the user's quotas on that file system must be in-core. This information is cached, so that not all information must be present in-core all the time. This is defined as (MAXUSERS * NMOUNT) / 4 + NPROC, where NMOUNT is the maximum number of mountable file systems.

Run-time Calculations

The most important data structure sized at run-time is the file system buffer cache. The system allocates 10% of each half-megabyte after the first half-megabyte to the cache. Thus on a 1 Megabyte machine, 50 kilobytes is allocated to the cache, while on a 2 Megabyte machine, 150 kilobytes is allocated to the cache. In any case, not less than 16 pages of file system buffers is allocated.

The number of buffers to be allocated can be forced to a specific value by patching the kernel variable nbuf with adb:

```
# adb -w /vmunix
nbuf?W 0t32
nbuf:    0 = 20
$q
#
```

sets the number of buffers to be 32 (decimal) independent of the amount of main memory available. Reboot after performing this series.

System Size Limitations

Because the file system block numbers are stored in page table pg_blkno entries, the maximum size of a file system is limited to 2^19 2048 byte blocks. Thus no file system can be larger than 1024M bytes.

The count of mountable file systems is limited to 15. This should be sufficient. If you have many disks it makes sense to make some of them single file systems, and the paging areas don't count in this total. To increase this, you must change the core-map (only if you have source) /sys/h/cmap.h, since there is a 4 bit field used here. The size of the core-map will then expand to 16 bytes per 2048 byte page. Don't forget to change MSWAPX and NMOUNT in /sys/h/param.h also.

The maximum value NOFILE (open files per process limit) can be raised to is 30 because of a bit field in the page table entry in /sys/machine/pte.h. 30 is the default.

6.5. System Log Configuration

Various system daemons and programs record information in the system log to aid in problem analysis. They send this information as Internet datagrams directed to the 'syslog' daemon on host 'loghost'. The daemon receives these datagrams and records the information or notifies users of problems. See `syslog`(8) for more details on this process.

The default configuration runs a `syslog` daemon on each machine, and also keeps all datagrams on the local machine (by maintaining the 'loghost' alias in the `/etc/hosts` entry on the local machine, or in the NFS environment, in the `/etc/hosts` entry exported by the `yp` master server machine for your domain). If you are running standalone, the default is for the `syslog` daemon to keep all datagrams on your machine. However, in a network environment it's much easier to track problems if all machines log their information in a single place. Therefore, during first time UNIX installation, the `setup` program reconfigures things so that only the designated server is the 'loghost': `setup` strips the 'loghost' alias from the `/etc/hosts` entries for the clients, and adds the alias for the server machine. This means that, for example, if the machine named krypton is your network server, the beginning of your machines' `/etc/hosts` files might look like:

```
192.9.1.1   krypton loghost
192.9.1.2   wally
192.9.1.3   beaver
192.9.1.4   june
192.9.1.5   eldridge
```

Now all datagrams sent to 'loghost' (from wally, beaver, etc.) are sent to krypton. There might also be other aliases on the same line of the `/etc/hosts` entry, like 'lprhost' or 'mailhost' — that's OK. Note also that, since the `syslog` daemon only starts up when messages must be handled, only the 'loghost' runs the daemon.

If you want to change this configuration — for example, if you have more than one server, and you want only one 'loghost' — simply change the placement of the 'loghost' alias, and then re-copy `/etc/hosts` to all machines. Test your system log configuration by running:

```
% tail -f /usr/spool/log/syslog
```

on the loghost machine, then sending any kind of mail on the various other machines. Each message sent will generate four or five lines of output if things are working.

6.6. Monitoring System Performance

The `vmstat` program provided with the system is designed to be an aid to monitoring systemwide activity — see `vmstat(8)`. Together with the `ps(1)` command (as in `ps av`), it can be used to investigate systemwide virtual memory activity. By running `vmstat(8)` when the system is active you can judge the system activity in several dimensions: job distribution, virtual memory load, paging and swapping activity, disk and cpu utilization. Ideally, there should be few blocked (b) jobs, there should be little paging or swapping activity, there should be available bandwidth on the disk devices (most single arms peak out at 30-35 tps in practice), and the user cpu utilization (us) should be high (above 60%).

If the system is busy, then the count of active jobs may be large, and several of these jobs may often be blocked (b). If the virtual memory is active, then the paging daemon will be running (sr will be non zero). It is healthy for the paging demon to free pages when the virtual memory gets active; it is triggered by the amount of free memory dropping below a threshold and increases its pace as free memory goes to zero.

If you run `vmstat(8)` when the system is busy (a 'vmstat 1' gives all the numbers computed by the system), you can find imbalances by noting abnormal job distributions. If many processes are blocked (b), then the disk subsystem is overloaded or imbalanced. If you have several non-DMA devices or open teletype lines that are 'ringing,' or user programs that are doing high-speed nonbuffered input/output, then the system time may go high (60-70% or higher). It is often possible to pin down the cause of high system time by looking to see if there is excessive context switching (cs), interrupt activity (in) or system call activity (sy).

If the system is heavily loaded, or if you have little memory for your load, then the system may be forced to swap. This is likely to be accompanied by a noticeable reduction in system performance and pregnant pauses when interactive jobs such as editors or window system programs swap out. If you expect to be in a memory-poor environment for an extended period you might consider administratively limiting system load.

The `nfsstat(1)` command displays statistical information about the Network File System (NFS), Remote Procedure Call (RPC), and Network Disk (`nd`) interfaces to the kernel. It can also be used to reinitialize this information.

If `nfsstat` with the '`-d`' flag (display `nd` information) shows a packet drop rate of more than 3%, you should suspect problems in the network interface hardware. Check the transceiver, the cable, and the ethernet board. See `nfsstat(1)` for details about the command's options.

6.7. Resource Control

Resource control in the current version of UNIX is rather primitive. The resources consumed by any single process can be voluntarily limited by the mechanisms of `setrlimit`(2). This can be done at the command level in `csh` by using the 'limit' command — see `csh`(1). Disk space usage can be monitored by `quot`(8) or `du`(1). No system-enforced procedure for controlling a user's disk space usage is implemented under the current system, although a modicum of control can be obtained by dividing user groups between different disk partitions.

6.8. Accounting

To run accounting, the system kernel must include the **`options SYSACCT`** and **`pseudo-device sysacct`** lines. Make sure your kernel has them if you are trying to make accounting work.

UNIX records two kinds of accounting information: connect-time accounting and process-resource accounting. Connect-time accounting information is stored in the `/usr/adm/wtmp` file, which is summarized by the program `/etc/ac` (see `ac`(8)). Process-time accounting information is stored in `/usr/adm/acct`, and analyzed and summarized by the program `/etc/sa` (see `sa`(8)).

If you need to charge for computing time, you can implement procedures based on the information provided by these commands. A convenient way to do this is to give commands to the clock daemon `/etc/cron` to be executed every day at a specified time. This is done by adding lines to `/usr/lib/crontab`; see `cron`(8) for details.

6.9. Making Local Modifications

Locally written commands are typically kept in `/usr/src/local` and their binaries in `/usr/local`. This allows `/usr/bin`, `/usr/ucb`, and `/bin` to correspond to the distribution tape (and to the system manuals). People wishing to use `/usr/local` commands should be made aware that they aren't in the base manual.

A `/usr/junk` directory to throw garbage into, as well as binary directories `/usr/old` and `/usr/new` are useful. The `man` command supports manual directories such as `/usr/man/manj` for junk and `/usr/man/manl` for local manual page entries to make this or something similar practical.

6.10. Files Needing Periodic Attention

The following files require periodic attention or may have been modified specifically for your system. They should be checked before upgrading and before being exported to other systems. Many of these files are mentioned in various sections of this Administrator's manual. You will find further details about them there, or on the appropriate page in the *Commands Reference Manual* or the *System Interface Manual*.

Table 6-1 *Files Needing Periodic Attention*

`/etc/fstab`	how disk partitions and file systems are used
`/etc/gettytab`	terminal configuration data base
`/etc/printcap`	printer data base
`/etc/remote`	names and phone numbers of remote machines for `tip(1)`
`/etc/group`	group memberships (served by `yp`)
`/etc/motd`	message of the day, printed at login
`/etc/passwd`	password file; each account has a line (served by `yp`)
`/etc/nd.local`	defines nd (soft) partitions
`/etc/rc.local`	local system restart script; runs reboot; starts daemons
`/etc/hosts`	host name data base (served by `yp`)
`/etc/networks`	network name data base (served by `yp`)
`/etc/services`	network services data base (served by `yp`)
`/etc/servers`	inet server data base
`/etc/hosts.equiv`	hosts under same administrative control
`/etc/securetty`	restricted list of ttys where root can log in
`/etc/ttys`	enables/disables ports
`/etc/ttytype`	terminal types connected to ports
`/usr/lib/crontab`	commands that are run periodically
`/usr/lib/aliases`	mail forwarding and distribution groups
`/usr/lib/sendmail.cf`	sendmail configuration file
`/usr/adm/acct`	raw process account data
`/usr/adm/messages`	system error log
`/usr/adm/shutdownlog`	log of system reboots
`/usr/adm/wtmp`	login session accounting
`/usr/spool/*`	spool directories for mail, printers, uucp, etc.
`/usr/spool/uucppublic`	holds files sent from other sites
`/.rhosts`	hosts with root login permission
`/dev/*`	device special files

7

Upgrading System Software

Upgrading System Software **217**

7.1. What To Save When Upgrading 218
7.2. How To Merge Old Files Into The New System 220

7

Upgrading System Software

This chapter outlines procedures for upgrading software to a new release. It tells you what files to save when upgrading, and explains how to merge old files into the new system.

7.1. What To Save When Upgrading

No matter what version of the system you may be running, you will have to rebuild your root and `/usr2` file systems. The easiest way to do this is to save the important files on your existing system, perform a bootstrap as if you were installing a software release on a brand new machine, then merge the saved files into the new system. The following lists the standard set of files you will want to save **(in addition to all user files)** and indicates directories in which site-specific files should be present. This list will probably be augmented with non-standard files you have added to your system. Do a `tar` of the directories important at your site, and include the directories `/etc/`, and `/usr/lib/` so that you don't miss anything the first time around.

Table 7-1 *Standard List of Files to Save when Upgrading*

`/.profile`	root sh startup script
`/.login`	root csh startup script
`/.cshrc`	root csh startup script
`/.rhosts`	list of hosts/users trusted at the superuser level
`/dev/MAKEDEV.local`	for the LOCAL case for making devices
`/etc/fstab`	disk configuration data
`/etc/gettytab`	tty port speeds database
`/etc/group`	group database
`/etc/hosts`	hosts database
`/etc/hosts.equiv`	list of trusted hosts on your network
`/etc/nd.local`	network disk local initialization file
`/etc/networks`	list of internet networks
`/etc/passwd`	user data base
`/etc/printcap`	printer capability database
`/etc/rc`	system startup file
`/etc/rc.local`	for any local additions
`/etc/remote`	remote hosts description database
`/etc/ttys`	terminal line configuration data
`/etc/ttytype`	terminal line to terminal type mapping data
`/etc/termcap`	for any local entries which may have been added
`/private`	system specific configuration files
`/usr/include/*`	for local subdirectory and any other additions
`/usr/lib/aliases`	mail forwarding data base
`/usr/lib/crontab`	cron daemon data base
`/usr/lib/font/*`	for locally developed font libraries
`/usr/suntool/fixedwidthfonts/*`	for locally developed window font libraries
`/usr/lib/tabset/*`	for locally developed tab setting files
`/usr/lib/tmac/*`	for locally developed troff/nroff macros
`/usr/lib/uucp/*`	for local uucp configuration files
`/usr/local`	for locally developed programs
`/usr/preserve`	editor temporary files saved here after crashes or hangups.
`/usr/spool/*`	for current mail, news, uucp files, etc.
`/usr/*`	all users' directories

You can save your system and user files by running the commands shown below while logged in as superuser. For *tape*, substitute the device abbreviation for

your tape controller as shown in Table 7-2. And for # substitute the correct unit number of the tape device.

Table 7-2 *UNIX Tape Device Abbreviations*

Abbreviation	*Device*
mt	Tapemaster half-inch magnetic tape
xt	Xylogics half-inch magnetic tape
st	SCSI quarter-inch tape
ar	Archive quarter-inch tape

For example, if you are using your first (or only) SCSI tape drive, enter `/dev/rst0`. If your system doesn't have a tape drive, you'll have to use slightly different commands, shown below, which access a tape drive elsewhere on the network. We'll call the machine with the tape drive your 'remote host'. Thus, before you begin, you'll have to know the device abbreviation for the remote host's tape drive (abbreviations are the same as in table above), and the remote host's hostname.

1. First, `tar` off the system files.

   For a machine with its own tape drive:

```
# cd /
# tar cf /dev/rtape# .??* dev/MAKEDEV.local etc lib usr/include usr/lib
```

   For a machine without a local tape drive, use the commands below. Substitute your remote host's hostname for *remote_host*. Use 126 for *block_size* on a quarter-inch tape, and 20 for a half-inch tape. Note that the command should be typed as a single line; be sure to escape the newline with a backslash (\), as shown, before you type <RETURN>:

```
# cd /
# tar cfb - block_size .??* dev/MAKEDEV.local etc lib usr/include\
usr/lib | rsh remote_host dd of=/dev/tape# obs=block_sizeb
```

   Either sequence makes a tape containing the system files you will need to set up your system after the upgrade.

2. NOW CHANGE TO ANOTHER BLANK TAPE, and run the following command. As described above, substitute the correct device specification for *tape#*. Also, replace *usera . . .usern* with the names of all users on your system. (You can check all names by looking in `/etc/passwd` or by doing `ls usr`.)

   For a machine with a tape drive:

```
# tar cf dev/rtape# usr/{spool,local,usera,userb,rest of users...}
```

For a machine without a tape drive (Note that the command should be typed as a single line; be sure to escape the newline with a backslash (\), as shown, before you type <RETURN>):

```
# tar cfb - block_size usr/{spool,local,usera,userb,rest of users...} | \
rsh remote_host dd  of=/dev/tape# obs=block_sizeb
```

This makes a tape containing users' files.

You may want to use other options on the *tar* commands, such as **b** to specify a large blocking factor like 126 for Archive quarter-inch tapes, or **v** to list each file as it is processed. See `tar`(1) for more information.

Once you have saved the appropriate files in a convenient format, the next step is to do a full (level 0) dump of all your file systems to tape with `/etc/dump` (see `dump`(8)).

After you complete the system dump, install the new release from the distribution tape as described in the chapter, *Installing UNIX on the Sun Workstation* (except that you may skip the disk-reformatting phase). If you are installing the release on a machine without a tape drive, consult the section, *Installing UNIX on Systems Without Tape Support.* Then proceed with the section below.

7.2. How To Merge Old Files Into The New System

When your system boots reliably and you have the root and `/usr2` file systems fully installed, you may proceed to the next step in the conversion process — merging your old files into the new system.

1. From the first `tar` tape you created in Step 1, extract the appropriate files into a temporary directory, say `/usr/convert`.

   For a machine with a tape drive:

```
# mkdir /usr/convert
# cd /usr/convert
# tar xfb /dev/rtape# block_size
```

   For a machine without a tape drive:

```
# mkdir /usr/convert
# cd /usr/convert
# rsh remote_host dd if=/dev/tape# bs=block_sizeb | tar xBf -
```

2. Certain files, such as those from the `/etc` directory, may simply be moved back into place:

```
# cp passwd group fstab ttys ttytype /etc
# cp crontab /usr/lib
```

Remember to remove the old copies after you're **sure** your system completeness/consistency is confirmed. If you wish, you can `mv`(1) the files instead of copying them.

Other files must be merged into the distributed versions by hand (`diff`(1) is often useful here). In particular, be careful with `/etc/termcap`.

3. The spooling and local directories, and the user files saved on the second `tar` tape may be restored in their eventual resting places without too much concern. Be sure to use the **p** option to `tar` so that files are recreated with the same file modes (you may want to add other options, as described in the previous section):

   For a machine with a drive:

```
# cd /
# tar xfbp /dev/rtape# block_size
```

   For a machine without a drive:

```
# cd /
# rsh remote_host dd if=/dev/tape# ibs=block_sizeb | tar xBfp -
```

4. Whatever else is left is likely to be site-specific or require careful scrutiny before you place it in its eventual resting place. Refer to the documentation (`hier`(7), for example) before arbitrarily overwriting a file.

8

Diag — A Disk Maintenance Program

This chapter describes `diag`, the Sun Microsystems standalone disk utility program. It describes how to start `diag`, how to prepare a new disk for operation, and how to fix problems on a disk.

8.1. Architecture

A disk consists of a stack of spinning platters, each with its own **head**. The heads, which all move together, travel back and forth across the surface of the platters between the outer rim of the stack and the center.

Each platter consists of a series of **tracks**, which are concentric rings that run 360 degrees around the platter. Each track is divided into **sectors**, and each sector is marked with a unique header. Sectors are the basic storage units; each sector contains 512 bytes of usable data. Sectors are also referred to as `blocks`.

A **cylinder** is a vertical stack of tracks; for example, the third track on all the platters in the stack is cylinder 3. Because the heads move together, they are always on the same cylinder.

Cylinder, Head and Sector Numbers

`Diag` understands disk addresses in two forms: cylinder/head/sector (CC/HH/SS), or absolute decimal block numbers (NNNNNN). `Diag` accepts either, but the CC/HH/SS form resembles the physical architecture of the disk more closely. Both forms identify one particular sector.

A CC/HH/SS address in the form CC/00/00 identifies a cylinder boundary; the cylinder starts at sector 0 in track 0. Similarly, each track starts at sector 0, so an address in the form CC/HH/00 identifies a track boundary.

Logical vs. Physical

An address can be logical or physical. A logical address represents a count of usable sectors from a partition boundary, and a physical address represents an actual location on the surface of the disk.

When UNIX reports disk errors, it provides the file system name and a logical block count, which is the number of usable sectors between the partition boundary and the starting sector of the transfer where the error occurred. That is, one of the sectors in a particular transfer had an error, but UNIX reported the address of the first sector in the transfer.

Partitions and File Systems

Partitions are regions on the disk set up by using the `partition` command. They start on cylinder boundaries and are used by UNIX.

`Diag` has built-in partition tables for all standard disk drive configurations plus a couple of alternates; you can select these easily using the `label` command. If these are not adequate, the `partition` command also has a facility for developing custom partitions.

SCSI Interface

The SCSI board is actually an adaptor which takes commands from the system bus and reissues them to the SCSI bus. The Adaptec controller on the SCSI bus receives these commands. It converts requests for absolute block numbers into cylinder/head/sector addresses, and takes care of the physical details. It communicates to the SCSI disk using an ST-506 interface.

All this insulates the CPU and file system from the physical details of SCSI disk geometry; it sees an idealized device consisting of a continuous stream of sectors.

SMD Interface

The CPU and file system must be more intimately acquainted with the details of the SMD disk; the SMD controllers are connected directly to the system bus. The CPU writes commands directly to the SMD control registers, and the controller carries out specified operations by communicating directly with the disk using the SMD interface. The device driver needs a lot of knowledge about the specific nature of the disk drive, and the filesystem takes advantage of this to arrange data on the disk in a way that will provide fast access for reading, writing, and updating files.

Removing Bad Sectors

All disk surfaces contain some media defects. The sectors where these occur must be removed from service to provide reliable operation.

Practically speaking, new defective sectors should never turn up during operation unless you have problems. All disks start life with a few surface defects, but the disk drive manufacturer and Sun each perform extensive surface analysis and slip or map these. Also, the procedure for preparing a new disk includes a third surface analysis to catch any surface defects caused during transit.

If you encounter surface defects after all that, it's usually caused by damage to the defective sector map, a "head kiss" where a head actually touches the surface of the disk, or an electronic glitch, which can corrupt the data on a sector making it look like a surface defect. Since these things usually indicate some underlying problem, if they happen once, they are likely to happen again. The disk troubleshooting section later in this chapter provides instructions for finding and fixing these problems

The following sections discuss defect handling on both SMD and SCSI disks.

SMD Bad Sectors

`Diag` commands deal with defective sectors on an SMD-controlled disk by `slipping` them or *mapping* them. `Format` and `fix` search a defined area for defective sectors. If they find one, they attempt to slip it and if that fails they map it. The `map` command simply maps sectors on request.

8

Diag — A Disk Maintenance Program

Diag — A Disk Maintenance Program **225**

8.1. Architecture 225
Cylinder, Head and Sector Numbers 225
Logical vs. Physical 225
Partitions and File Systems 226
SCSI Interface 226
SMD Interface 226
Removing Bad Sectors 226
SMD Bad Sectors 226
SCSI Bad Sectors 227
8.2. Starting Diag 227
User Interface 227
Booting `diag` 228
Configuring *diag* 229
8.3. Preparing a New Disk 233
SCSI Disks 234
Using fix to Format SMD Disks 237
8.4. Troubleshooting With Diag 240
Checking and Fixing a Bad Sector (SCSI) 241
Checking and Fixing a Bad Sector (SMD) 243
Electronic Problems 245
8.5. Command List 246
Toggle Flags and Options 246

Miscellaneous Commands 247
Tests 247
Complicated, Interactive Commands 248

CAUTION **Do not use `format` on any SMD disk purchased from Sun or you will erase valuable mapping information.**

Disks configured for sector slipping contain a spare data sector at the end of each track. When `diag` attempts to slip a sector, it removes the defective sector from service by identifying it with a special header, then it bumps subsequent logical sectors forward until the last sector moves to where the spare was.

If the disk is not configured for sector slipping, or if a track contains a second bad sector, you must rely of mapping. `Diag` writes the defective sector's address in the bad sector map and marks the sector with a header identifying it as a mapped sector. When the access to the mapped sector fails, the bad sector map redirects UNIX to use another sector on the spares track.

Sector slipping improves disk efficiency; it allows the disk to operate without the additional overhead of mapping.

To identify slipped sectors, mapped sectors, and to tell if the disk is configured for sector slipping, use the `rhdr` command, described later in this chapter.

SCSI Bad Sectors

To deal with SCSI disk bad sectors, the controller marks the bad sector and resumes its sector count on the next sector. It then slips the physical locations of all subsequent sectors towards the end of the disk.

As defects add up, the sectors get skewed across cylinder and track boundaries. To deal with the difference between where a sector should be and where it has slipped to, the controller does a quick scan of the disk when it comes up. It divides the disk into zones and keeps a table of the amount of slippage for each zone. When it does a seek, it adds the amount of slippage for each zone it has to go through, so that it ends up looking somewhere near the desired sector.

Because slipping requires moving all subsequent sectors on the disk, all of diag's strategies for dealing with defective sectors involve reformatting the disk. When it's finished formatting and slipping sectors, it writes a list of the slipped sectors at the end of the disk.

8.2. Starting Diag

`Diag` is a standalone program that lives in `stand/diag`. You must boot it standalone (without UNIX) from the system monitor. After booting, you must then configure `diag` to match your controller/disk configuration. This section provides instructions for both.

User Interface

`diag` responds to commands typed on the workstation keyboard. It contains a number of subsystems, where the command sets are different, and it changes its prompt to remind you of this. The normal prompt is:

```
diag>
```

When you are in the `format` subsystem, it changes to:

```
format>
```

to remind you where you are.

A couple of tricks make `diag` easier to use:

Restart
: If you exit from `diag` to the monitor, you can restart it with the monitor command:

```
> g4000
```

Command Shortening
: You can shorten most `diag` commands to a single letter, or to the shortest unique string of letters. For example, to enter the `translate` command, just type:

```
diag> t
```

Booting `diag`

`Diag` boots from the PROM monitor's command interpreter. To activate the command interpreter, power-on or reset the system. If UNIX is active, power it down properly, as described in the "Sun System Administration Manual" before resetting the system. When the message:

```
Auto-boot in progress
```

appears, press either:

L1 -a (while holding down "L1", press "a") from the keyboard, **or**

BREAK (press the "break" key) from a terminal.

The monitor should respond by interrupting the UNIX bootstrap and displaying a "greater than" symbol, which is its prompt.

```
>
```

To boot `diag` from the monitor, enter:

```
>b path_name
```

where *path_name* is the path to, and the name of the file to be booted. From a formatted, operational disk, use:

```
>b stand/diag
Boot: disc(0,0,0)stand/diag
Load: disc(0,0,0)boot
Boot: disc(0,0,0)stand/diag
Size: 34816+20480+1160 bytes          [varies by version]

where disc =
xy for a Xylogics controller,
sd for a SCSI controller,
ip for an Interphase controller,[1] or
ec for 3Com, ie for Sun ethernet boards or le
  for Lance ethernet boards.
```

Booting `diag` from a tape requires loading the boot block from the tape, then loading `diag` from the tape. Normally `diag` is the file immediately after the boot block on the tape (file number 1).

To do this, type:

```
>b tape()
Boot: tape(0,0,0)
Boot: tape(0,0,1)
Size: 34816+20480+1160 bytes          [varies by version]
```

where *tape* is **mt** for 1/2" tape with Tapemaster controller, **xt** for 1/2" tape with Xylogics controller, **ar** for 1/4" Archive tape, and **st** for 1/4" SCSI tape.

When `diag` first starts up, it displays a sign on message:

```
Version 2.2 84/08/10                  [varies with different versions]
Disk Initialization and Diagnosis

When asked if you are sure, respond with 'y' or 'Y'
```

Earlier versions of `diag` may not display the version message. These are outdated; use a newer version.

Configuring *diag*

When `diag` starts, it automatically enters the `diag` command subsystem. It prompts for required hardware-specific configuration information, then returns to the command level. To change the configuration later from the command level, enter the command `diag`.

`Diag` needs detailed information about the disk and controller it is working with. It has information about standard configurations built in, and it has a facility for

[1] Sun Microsystems no longer officially supports Interphase 2180 controllers.

accepting information about a non-standard disk and controller combination.

The configuration procedure differs, depending on whether it's being configured for a standard SMD or SCSI interface, or something non-standard.

In the following command descriptions, where the example or explanation applies to both types of disk, it clearly says so. Ignore sections which describe a different interface type.

Configuring for Standard Disks

First, `diag` asks what type of disk controller it's dealing with:

```
specify controller:
         0 - Interphase SMD-2180
         1 - Xylogics 440 (prom set 926)[2]
         2 - Xylogics 450
         3 - Adaptec ACB 4000 - SCSI
which one? 2
```

This example shows a Xylogics 450 (type 2) disk controller.

Next, `diag` asks for the controller's bus address. After you provide an address (see the table below for defaults) it echoes the address back:

```
Specify controller bus address in hex: address from table
Device address: address you selected
```

Table 8-1 *Controller Bus Addresses*

Controller type	MULTIBUS ADDRESS		VMEbus ADDRESS	
	1st Cntrlr	2nd Cntrlr	1st Cntrlr	2nd Cntrlr
Xylogics	ee40	ee48	ee40	ee48
Adaptec (Sun-2)	80000	84000	ee2800	N/A
Adaptec (Sun3)	N/A	N/A	200000	N/A

If you specify a controller which interfaces to a SCSI disk, `diag` then asks for the controller's SCSI bus unit number:

```
Which target? 0
```

0 is the target number for the first (or only) SCSI disk controller; **1** is the target number for the second.

Next `diag` requests the physical unit number of the disk on the controller:

```
Which unit? 0
```

[2] Sun Microsystems no longer officially supports Xylogics 440 controllers.

0 is the correct response for the first (or only) disk drive connected to the selected controller; **1** is the correct response for the second, and so on. SMD disks can be set for unit numbers of 0 to 3 while SCSI disks are either unit 0 or unit 1.

Next `diag` displays a menu of the different disks for which it has built-in configuration information, then asks for the disk drive type. If you select one of these (except for `other`), `diag` prints some physical data about that disk, including the number of cylinders, number of alternate cylinders, number of heads, and number of sectors per track. If you select `other`, it prompts for this information. Then it initializes the controller, issues a `status` command to the device, and displays results. This works like the `status` command issued to the `diag>` prompt.

The following example shows a Fujitsu M2312K (84–Mbyte unformated) disk controlled by a Xylogics 450 controller with Revision "C" PROMs.

```
Specify drive:
          0 - Fujitsu-M2312K
          1 - Fujitsu-M2284/M2322
          2 - Fujitsu-M2351 Eagle
          3 - Other
which one? 0
ncyl 586 acyl 3 nhead 7 nsect 32
status: ready
drive status: ready
Xylogics PROM Rev 'C'
```

If you have an older Xylogics board with Rev A PROMs, `diag` issues a warning message stating that these boards should be upgraded. They perform certain operations incorrectly, which causes the software to be unable to detect some ECC errors.

Now `diag` has the information it needs about the disk. It returns to the command mode and displays its prompt.

```
diag>
```

If the sequence fails before this point, check the hardware cabling and the information already given. Then use the `diag` command to reenter the data, or reboot `diag`.

Other Disks

If you select the *other* drive type `diag` asks for information about the drive. Once you have defined the new drive type, `diag` adds this entry to the list of drive types, enabling you to use it over and over. `Diag` allows you to define a total of 4 `other` drive types.

NOTE *Consult the disk drive manual for information about* `other` *disk drives.*

The following two examples show how to configure an `other` disk drive for an SMD, then a SCSI controller.

The first example shows the questions asked when using a Xylogics 450 controller for a imaginary SMD disk called a "Bogus-M1234". The second example shows the same for an imaginary SCSI disk called a "Phony-4321".

Imagine that the manufacturer's manual says that the Bogus 1234 has 823 cylinders, 10 heads, and 32 sectors/track.

```
Specify drive:
         0 - Fujitsu-M2312K
         1 - Fujitsu-M2284/M2322
         2 - Fujitsu-M2351 Eagle
         3 - Other
which one? 3
# of data cylinders? 820                          [total # of cyls minus alts]
# of alternate cylinders? 3                      [cyl for bad sec forwarding]
first head? (usually 0, 2 for Lark fixed) 0
physical partition? (usually 0, 1 for Lark cartridge) 0
# of heads? 10
drive type? 3                                     [Xylogics only – # 0 to 3]
ASCII identification? Bogus-M1234                 [Used for labeling disk]
                                                  [same as partition command]
# of sectors? 32                                 [sectors per track]
ncyl 820 acyl 3 nhead 10 nsect 32
status: ready
drive status: ready
Xylogics PROM Rev 'C'
```

NOTE *On the Xylogics 450 controller, all disks with the same drive type must be the same type of drive. This is straightforward in the above example; the Bogus-M1234 is assigned drive type 3. To add a second "other" different from the Bogus-M1234 you would have to assign it the drive type of one of the existing configurations. If you do this, be sure you do not assign it the drive type of a disk that is or will be connected to that controller. For more on drive type numbers, see the command 'rhdr'.*

`Diag` asks different questions for a SCSI controller. The following example shows how to configure a Phony-4321 with 578 cylinders, 5 heads, and 17 sectors per track. The drive manual recommended using a buffered seek of 2 and write precomp starting at cylinder 0.

```
Specify drive:
          0 - Micropolis-1304
          1 - Micropolis-1325
          2 - Maxtor-1050
          3 - Fujitsu-2243AS
          4 - Vertex-185
          5 - Other
which one? 5
# of data cylinders? 576                    [total minus alternates]
# of alternate cylinders? 2                 [bad sector spaces]
buffered seek? (usually 2) 2                [from drive manual]
cyl # to start write precomp? 0             [from drive manual]
# of heads? 5
ASCII identification? Phony-4321            [For labeling disk]
# of sectors? 17                           [sectors per track]
ncyl 576 acyl 2 nhead 5 nsect 17
status: 6 Word mode Dma ena
```

8.3. Preparing a New Disk

This section describes how to use `diag` to prepare new SCSI and SMD disks for operation These discussions assume that `diag` is loaded and configured as described earlier.

For SCSI disks, this process includes formatting the disk, performing surface analysis, then writing a label on the disk. The format divides the disk into blocks and sectors, the surface analysis checks for and repairs surface defects, and the label writes important information on the disk, including the disk name, and the partition map.

For SMD disks, this process includes using `fix` to perform surface analysis, then writing a label on the disk.

CAUTION **Do not format any SMD disk you receive from Sun. This will destroy mapping information that you cannot replace. To perform the surface analysis of your SMD disk, use the `fix` command in the SMD**

`format` subsystem, as described later.

The partition table defines the disk partition boundaries, which UNIX uses for file system boundaries. You should have a partition table ready to write when you start this procedure. If you are going to use one of `diag`'s built-in partition tables, this is no problem, but if you are going to use any other partition table, see the instructions for the `partition` command later in this chapter.

NOTE *If you are installing a new system, we recommend that you use the instructions in Installing UNIX on the Sun Workstation to prepare your disks.*

`diag` provides 2 different format subsystems; one for SCSI disks and one for SMD. This section provides separate sections for each.

SCSI Disks

The following procedure provides instructions for testing, formatting, and labeling a SCSI disk. Use the following procedure to format and label a disk:

1. Load and boot `diag`.
2. Type **`format`** to access `diag`'s SCSI format subprogram (`diag` knows it's dealing with a SCSI disk from information provided during configuration). It displays its prompt:

```
diag> format
SCSI format.
format>
```

To see a list of the SCSI format subcommands, type:

```
format> ?

SCSI Format Subcommands:
   f: format disk
   r: read defect list from disk
   p: print defect list from disk
   a: add defect to list
   d: delete defect from list
   s: surface analysis
   t: translate block no. to cyl/hd/bfi
   q: quit format
```

3. Check to make sure that the disk defect list written on the disk matches the hardcopy list shipped with your drive. Enter the command `r` to read the list from the disk into RAM, then enter the command `p` to display it on the screen:

CAUTION **To avoid erasing valuable bad sector information, use the commands `r` and `p` in the correct sequence. The command `r` writes the bad sector list from the disk to RAM; the command `p` displays the bad sector list currently in RAM on the screen. To write the list from RAM to disk, you must format the disk using the command `f`. If you do a format without having done an `r`, you will overwrite the list that was on the disk. If you do an `r` without having done a format, you will overwrite any corrections made by either you or the format subsystem.**

NOTE *The location of the hardcopy list depends on the workstation type. It is usually taped to the front or top of the pedestal. If you don't see it or have misplaced it during unpacking, look for a second copy taped to the disk drive housing inside the pedestal; to find it, consult the appropriate hardware manual.* **Be sure to replace this second copy when you have used it.** *Also note that on a some drives, the hardcopy list groups defects by head; the cylinder and bytes from index numbers appear in the CYL and BI columns. On others, the numbers appear in the CYL, H, and BYTE columns respectively.*

```
format> r
format> p
Defect list
Defect  Cylinder  Head   Bytes from Index
  N         NN        N          NNNN
etc.
```

4. Make sure this either matches the hardcopy disk defect list. It may have a few more defects on it, but not fewer.

   a) If they match, continue with step 4.

   b) If the hardcopy list shows defects that are not displayed on the screen list, use the a command to add to the list in RAM (it will get written on the disk when you format it):

```
format> a
cylinder? number
head? number
bytes from index? number
```

   c) If you cannot read the list from the disk, use the a command as shown above to type in the entire hardcopy list.

   d) After making any changes to the list on the disk, use the p command to display the changes on the screen. Check again to make sure the copy on the screen matches the hardcopy list.

5. When you have verified the defect listing, format the disk with the format (f) command. After you type this command, the system displays a warning, then asks for confirmation:

```
format> format
format/verify, DESTROYS ALL DISK DATA!
are you sure? y
```

The formatting process takes three minutes or more. At the end, you should see a message:

```
Defect list written on disk
```

If you see any other message (SCSI reset, for example), the formatting process did not succeed, and the defect list was not recorded on the disk. **You must format the disk again.**

6. When you have successfully formated the disk, do a surface analysis using the (s) sub-command. This analyzes the entire disk surface, then displays a list of any defective sectors it found. For a new disk, you should ask for five

surface analysis passes:

```
format> s
# of surface analysis passes (5 is usual)? 5
```

Five passes may take an hour or more to complete. When it's done, it displays a message to announce that fact.

7. If it finds any bad sectors, it displays a report describing what it found, then adds the sector to the bad sector list in RAM. When its done, it tells you to re-format the disk:

```
Surface analysis complete
some_number bad sectors found
Use the 'f' command to format the disk.
format>
```

Before continuing, you **must**:

a. Reformat the disk (go back to step 4).

b. Continue this loop until the surface analysis reports that it found no bad sectors:

Once the surface analysis completes without finding any bad sectors, the format is successful.

8. Enter **q** to exit the format subsystem:

```
format> q
diag>
```

NOTE *The following 'label' operation writes a partition map on your disk. If you do not have a custom partition made, or you do not intend to use the default, you may continue, but note that you will change the partition map later.*

9. Type **label** to the diag prompt.

```
diag> label
```

10. diag now asks if you want to use the built-in partition map, then asks for confirmation:

```
diag> label
label this disk...
OK to use logical partition map 'your disk type'? y
Are you sure you want to write? y
```

11. After labeling the disk, `diag` automatically verifies the label it has just written. The following example shows the verify for a Fujitsu M2322:

```
[ This is an example only; do not enter this information. ]
verify label
id: <Fujitsu-M2322 cyl 821 alt 2 hd 10 sec 32 interleave 1>
        Partition a: starting cyl=0, # blocks=15884 [#'s vary]
        Partition b: starting cyl=50, # blocks=33440
        Partition c: starting cyl=0, # blocks=262720
        Partition g: starting cyl=155, # blocks=213120
diag>
```

12. If you confirm, `diag` partitions your disk according to the default maps shown below, then exits:

```
diag>
```

The following table shows the default partitions for Sun-supplied SCSI disks[3]:

Table 8-2 *Default Partition Sizes for SCSI Disk Subsystems*

SCSI Disk	*Raw*	*Partition Sizes (MBytes)*							
		"a"	"b"	"c"	"d"	"e"	"f"	"g"	"h"
Micropolis 1304	50	8.1	8.4	43.1	*unused*	*unused*	*unused*	26.5	*unused*
Micropolis 1325	85	8.1	17.1	70.9	*unused*	*unused*	*unused*	45.6	*unused*
Maxtor XT-1050	50	8.1	8.4	44.4	*unused*	*unused*	*unused*	27.9	*unused*
Fujitsu M2243AS	86	8.1	17.1	70.8	*unused*	*unused*	*unused*	45.6	*unused*
Vertex V185	85	8.1	17.1	70.9	*unused*	*unused*	*unused*	45.5	*unused*

Using fix to Format SMD Disks

CAUTION **Using `format` destroys the slipping and mapping information on the SMD disk. Since you cannot replace this information, do not use `format` on any SMD disk unless you are SURE you wish to erase this information.**

This section provides instructions for preparing formatting, checking, and labeling an SMD disk. It includes two paths for formatting and testing; one for disks purchased from Sun, and one for disks not purchased from Sun. If you got your disk from Sun, use Step 1A; it uses the `fix` command to check the surface and

[3] Note that the numbers in this table are approximate: formatted capacity depends on the type of controller. Also, note that a 'Megabyte' of disk capacity is defined as one million bytes, and that UNIX file storage capacity is substantially smaller.

repair defects, without losing any previous mappings and slippings (we added some at Sun). **It also preserves the existing bad sector map, adding to it as necessary.** If you got the disk from somewhere other than Sun, or if the format on the disk has become unusable for some reason, use Step 1B. This re-formats the disk, and finds and repairs defects, but it also overwrites the existing bad sector map, erasing the results of any previous testing.

NOTE *You must have a partition map ready before beginning this procedure.*

1 - A

This step retains the mapping and slipping information currently on the disk; use it if you purchased your disk from Sun, or if you wish to preserve the mapping and slipping information on it. **If you purchased your disk somewhere else, or if you wish to overwrite the mapping and slipping information on it, skip to step 1 - B - a.**

a. To get information about the disk, enter the `verify` command:

```
diag> verify
verify label
id: disk type
    Partition a: starting cyl=0, # of blocks=NN
    and so on...
diag>
```

b. Next, tell `diag` to `fix` the entire disk. Look at the display above, and add the number of data cylinders to the number of alternate cylinders. Then, enter the `fix` command, using the number obtained above in place of *nnn*:

```
diag> fix
fix -- DESTROYS SOME DISK DATA
formats a range of tracks
enter track number as 'cyl/track'
starting track? 0/0
ending track? nnn/0/0
# of surface analysis passes (5 recommended)? 5
OK to format from 0/0/0 to nnn-1/X/X ? y
NNN/NN
diag>
```

This process takes a while.

As `fix` formats each sector, it displays the cylinder and track number (represented by NNN/NN in the above display). Before `fix` exits, it updates the bad sector map. Note that if it is interrupted, it does not update the map, and you must repeat this process.

1 - B

This step overwrites any mapping and slipping information on your disk; Use it if you *did not* purchase your disk from Sun or if you wish to overwrite

the mapping and slipping information on your disk.

a. Enter the `format` command. `Diag` warns you that this destroys disk data, then asks for confirmation:

```
diag> format
format/verify.  DESTROYS ALL DISK DATA
are you sure? y
```

b. Next, `diag` asks for the number of surface analysis passes. We recommend a minimum of 5 surface analysis passes:

```
# of surface analysis passes (5 recommended)? 5
```

c. The rest is automatic: `diag` tests the surface of the disk, slips or maps any defective sectors it finds, then formats the entire disk. During the format, it displays the number of each cylinder it formats. When it's done, it displays a message:

```
cyl nnn format complete - 0 bad sector
Use the label command to label the disk
diag>
```

d. Continue on to Step 2.

NOTE *From here on, this procedure assumes you have are either going to use a default partition table or you have written or modified one using the* `partition` *command. The default partitions are designed for standalone use only; if you are installing an NFS server, or have any other reason for not using the default partition, construct or modify a partition with the* `partition` *command, then return here.*

2. Enter the command `label`:

```
diag> label
```

`Diag` now asks if you want to use default partition map, then asks for confirmation before proceeding:

```
diag> label
label this disk...
OK to use logical partition map 'disk type'? y
Are you sure you want to write?  y
```

NOTE `diag` *remembers if you recently used the partition command; if so, it uses the last partition you accessed in place of 'disk type' above. If you just created a*

custom partition called 'xyz', it asks:

```
OK to use logical partition map xyz?
```

3. After labeling the disk, `diag` automatically verifies the label it has just written. For example, the verify for a Fujitsu M2322 might look like this:

```
[ This is an example only; do not enter this information. ]
verify label
id: <Fujitsu-M2322 cyl 821 alt 2 hd 10 sec 32 interleave 1>
        Partition a: starting cyl=0, # blocks=15884
        Partition b: starting cyl=50, # blocks=33440
        Partition c: starting cyl=0, # blocks=262720
        Partition g: starting cyl=155, # blocks=213120
diag>
```

The following table shows the default partitions[4]:

Table 8-3 *Default Partition Sizes for SMD Disk Subsystems*

SMD Disk	*Raw*	*Partition Sizes (MBytes)*							
		"a"	"b"	"c"	"d"	"e"	"f"	"g"	"h"
Fujitsu 2312K (8")	84	8.1	17.1	67.3	*unused*	*unused*	*unused*	42.0	*unused*
Fujitsu 2284 (14")	169	8.1	17.1	134.5	*unused*	*unused*	*unused*	109.1	*unused*
Fujitsu 2322 (8")	168	8.1	17.1	134.5	*unused*	*unused*	*unused*	109.1	*unused*
Fujitsu 2351 Eagle	474	8.1	17.1	395.7	*unused*	*unused*	*unused*	369.8	*unused*

8.4. Troubleshooting With Diag

This section provides methods of using `diag` to deal with disk problems.

Usually, you find out you have disk problems when you see a UNIX error report. This should provide the file system name, and a disk block number. For example:

```
Sd0g read error block nnnn
```

To deal with this error, you must discover and repair the cause of the error and repair the surface of the disk.

This section provides the following procedures:

Fixing a Bad Sector (SCSI) — Describes how to use the information in a UNIX error report to add a sector to the bad sector list. (SCSI only).

[4] Note that the numbers in this table are approximate: formatted capacity depends on the type of controller. Also, note that a 'Megabyte' of disk capacity is defined as one million bytes.

Checking and Fixing a Bad Sector (SMD) — Describes how to go to the sector indicated by a UNIX error report, determine correct action, and repair the defect. This procedure provides a reasonable chance of salvaging most of the data on the defective sector.

Electronic Problems — Describes how to set up `diag` to test the swapping area of the disk, so you can perform electronic troubleshooting while using `diag` to check the disk functionality without destroying critical data on the disk.

Checking and Fixing a Bad Sector (SCSI)

Use this procedure to add a bad sector reported by UNIX to the bad sector list on the disk.

NOTE *Because of the nature of the SCSI interface, repairing any surface defect requires reformatting the disk, which erases all the data on it. To ensure the safety of your data, take a full dump before repairing any SCSI disk repair.*

1. Take full dumps of your entire system.
2. Boot and configure `diag` as described earlier in this chapter.
3. Use the `verify` command to find the number of the first cylinder on the partition having the error. Note that in the error report, the last letter in the device name identifies the partition.

   For example:

```
diag> verify
verify label
id: <Micropolis 1304 cyl 825 alt 5 hd 6 sec 177
  Partition a: Starting cyl = 0, # of blocks = 15884
  Partition b: Starting cyl = 156, # of blocks = 16422
  Partition c: Starting cyl = 0, # of blocks = 84150
  Partition g: Starting cyl = 317, # of blocks = 51816
```

4. Using the `add` command, add the block number reported by UNIX to the first cylinder of the partition. For example, if UNIX reported an error in `sd0g`:

```
diag> +
Number: NNN/      [Starting cylinder of g]
  plus: nnnn      [The block number]
= nnnnn = 0xnnnn = CCC/HH/SS (logical)
```

5. Do a `read` of the track to obtain the exact address of the failed block:

```
diag> read
starting block? NNN/NN/00      [End with 00]
# of blocks? 16
increment? 1
# of blocks per transfer? 1
NNN/NN/NN

And if it fails...
NNN/NN/NN Read failed, cyl=NNN, head = NN,
sector = NN scb: N check
[A few lines of other information]
```

6. Enter the `format` subsystem:

```
diag> format
SCSI format.
format>
```

7. Use the format subsystem `read` command to load the bad sector list into RAM.

CAUTION **DO NOT SKIP THIS STEP OR YOU WILL DESTROY YOUR BAD SECTOR MAP!**

```
format> r
```

8. Use the translate command to obtain bytes from index:

```
format> t
logical block number?  NNN/NN/NN
cyl NNN  head NN bfi NNNN (physical)
```

9. Use the format subsystem `add` command to add this data to the list:

```
format> a
cylinder? NNN
head? NN
bytes from index? NNNN  [use data from above!]
```

10. Now format the disk:

```
format> format
DISK FORMAT - DESTROYS ALL DISK DATA!
are you sure? y
Formatting... done              [Takes a while]
Defect list written on disk.
format>
```

11. Restore the system from the dump tape made earlier.

Checking and Fixing a Bad Sector (SMD)

To repair a bad sector on an SMD disk, use the following procedure:

1. Obtain the file system and block number from UNIX error message. Remember the last letter of the *device* identifies the partition.
2. Boot and configure `diag` as described earlier.
3. Using the block number and file system name provided by the UNIX error message, calculate the CC/HH/SS address of the defect as follows:

   a) Enter the `verify` command:

```
diag> v
verify label                    This is an example only
id: <Fujitsu-M2351 Eagle cyl 840 alt 2 hd 20 sec 46>
partition a: starting cyl # 0       # blocks 15884
partition b: starting cyl # 18      # blocks 16442
partition c: starting cyl # 0       # blocks 772800
parititon g: starting cyl # 55      # blocks 722200
```

   b) Use the `add` command to add the starting cylinder number of the file system with the error to the offset provided by the UNIX error message:

```
diag> +             Start addition
number? CC/00/00    Cyl boundary (head and sector = 0)
plus: nnnnnn        Add the offset to it
=dec =hex =CC/HH/SS     Finally, the answer
diag>
```

4. Using the CC/HH/SS results of the above calculation, do a `read` to test the entire track containing the suspected sector (remember, SS = 00). Tell it to test 16 tracks, using one block per transfer:

NOTE *It's a good idea to toggle the 'errors' flag so you can see retrys. Sometimes it is the only way to spot the error. To toggle the flag, enter:*

```
diag> e.
```

```
diag> read
read
Starting block? CC/HH/00
number of blocks? 16
Increment? 1
# of blocks per transfer? 1
CC/HH/NN diag>

Or, if you have an error:
Read failed #6, CRC/hard ECC error, cyl=nn head=nn sector=dd

Note: above message is example only. If you have 'errors' flag
set for retrys, a similar message man appear several times.
```

5. If your disk is configured for slipping, attempt to slip the bad sector:

```
diag> slip
slip sector
slipping may be removed only by complete format of the disk
cylinder number? NN
track number? NNN
logical sector to be slipped? NN
Attempt to preserve data? y
OK to attempt slip of logical sector CC/TT/SS? y
diag>
```

6. If the slip does not succeed, the following buffer shuffle gives you a fairly good chance to save your data:

   a) Use the `map` command as if to *map* the defective sector. When it asks if you want to save the data, say "yes":

```
Attempt to save data? y
```

   b) When it asks if you want to actually write in the mapping, say "no":

```
OK to map this sector? n
```

   c) Now do a `write` to the defective sector. Note that because `diag` still has the "saved" data in its buffers, it will restore the original data to the bad sector.

CAUTION **To avoid losing data, be sure to write to only the one sector that you used the `map` command on.**

```
diag> write
write
starting block? NN
# of blocks? 1
increment? 1
# of blocks per transfer? 1
CC/HH/SS diag>
```

7. Now repeat the `read` from above:

```
diag> read
read
Starting block? CC/HH/00
number of blocks? 16
Increment? 1
# of blocks per transfer? 1
CC/HH/NN diag>
```

8. If the sector checks out OK, the problem is solved. You may want to repeat the `read` a few times to make sure the sector is not marginal. If the read still reports an error, repeat the `map` done earlier, only when it asks for permission to map the sector, answer **`yes`**.

Electronic Problems

Many disk errors are caused by problems with the disk cables or the electronics. These problems tend to generate multiple disk error messages at random locations.

The following procedure shows how to set up `diag` to "bang" on the swap area of the disk. This is where the most expendable data lives; destroying the data here should cause minimum harm. This procedure does not describe how to troubleshoot the electronics; it only describes how to set up `diag` so that the electronics troubleshooting causes the least damage.

For electronics troubleshooting instructions, see the appropriate field service manual.

1. Start up and configure `diag` for the disk with the problem.
2. Enter the `verify` command. For a Micropolis 1304, the display goes:

```
diag> v
verify label
Micropolis 1304 cyl = 824, alt 5, hd 6, sec 17>
partition a: starting cyl # 0      # blocks 15884
partition b: starting cyl # 156    # blocks 16442
partition d: starting cyl # 0      # blocks 84150
parititon g: starting cyl # 317    # blocks 51856
```

On almost all disks, the swap space is *partition b*. To make `diag` do reads within partition b, request a `read` using a cylinder within *partition b*. For example, enter the following:

CAUTION **To avoid damaging data, make sure you really hit the swap space. Be sure a) you don't leave the slash off the cylinder number, b) the starting cylinder is within partition b, and c) the number of blocks does not lead into the next partition (g in this example).**

```
diag> r                    [This is an example only]
read
starting block ? 156/          [Don't forget the "/"!]
# of blocks? 16442         [Or fewer blocks to be safe]
increment ? 1
# of blocks per transfer ? 1
NNN/NN/NN
```

NOTE *In this example,* `diag` *will do reads to the entire swap space block by block. If you wish to make it read the same block over and over, use an increment of 0.*

8.5. Command List

The following list shows the `diag` subsystems, commands, and subcommands. Note that subsystems provide access to subcommands, while commands initiate action from the top layer of `diag`. Subprograms are routines called by subsystems.

For convenience, the commands are divided into several categories. These are:

Toggle flags and options
Miscellaneous commands
Perform some test
Do something complicated and interactive to the disk.

Toggle Flags and Options

All the flags and option toggles work similarly; the option is either ON or OFF, and calling the following commands cause it to switch states and display its new value:

`error` — When ON, displays messages for every retry; when OFF, displays a message only after all retries are done. (Most operations retry 3 times; formatting retries 1 time). (default = OFF).

`info` — When ON, provides verbose messages for every operation performed (default = OFF).

`time` — When ON, displays timing messages for formatting, read, and write operations (default = OFF).

`slipmsgs` — When ON, provides before and after dumps of track headers when a sector is slipped (default = OFF).

`mapcheck` — When ON, if an error occurs during a `read, write, position` or `test` command with a SMD disk, it reads the map table from

the disk, and checks to see if any sector in the range tested has been mapped. If so, it reports this fact. Since `diag` reports errors when trying to read a mapped sector, the mapcheck message may explain an otherwise mysterious error report (default = ON).

Miscellaneous Commands

Use the following commands to help you use `diag`:

`help` or ? — Display current list of commands available.

`quit` — Exit `diag`.

`clear` — Clear drive faults.

`status` — Fetch and display current controller and drive status.

`diag` — Reset configuration information (described earlier in this manual).

`translate` — Take a disk address and display it in CC/HH/SS, decimal, and hex, translated for the currently configured disk.

addition and subtraction — Entering a + or − on the command line causes a prompt asking you for two numbers (to be added or subtracted). `Diag` provides the result in decimal, hex, and cylinder/sector/head form, translated for the currently configured disk.

`version` — Displays all the *sccs* identifiers in the program.

`verify` — Reads the labels on the disk, prints out the partition map, and if necessary, asks if you want to restore the primary label.

Tests

The following commands perform non-interactive tests:

`position` — This non-destructive test checks the ability to seek and read sectors. It selects and reads single sectors at random, continuing until the user types ^C. If it encounters an error, if `mapcheck` is *on*, it checks to see if the sector is mapped; if so, it reports that fact.

`test` — This **destructive** test writes, then reads groups of sectors continuously, testing for ability to seek, write, and read. It selects sectors at random, and tests a block of sectors starting there. It prompts for the size of the block, and it continues until the user enters ^C. If it finds an error, and if `mapcheck` is ON, it checks the map to see if any sectors in the group are mapped. If so, it reports that fact.

`seek` — This non-destructive test does an hourglass seek over all cylinders, then reports the time it took.

`read/write` — This test checks for ability to read and write data. Write is **destructive** but read is not.

`dmatest` (Xylogics 450 controller only) — This test checks the Xylogics 450 controller using BUFLOAD and BUFDUMP commands. If `abortdma` is ON, it exits when it finds an error; otherwise it continues until the user types ^C.

Complicated, Interactive Commands

The following commands provide significant interaction with the disk. They are described in detail, later.

`map` (SMD disks only). This command enables you to read the current list of mapped sectors, and to (optionally) add a new sector to that list.

`fix` (SMD only). This command reformats and verifies a given range of the disk.

`slip` (Xylogics 450 only). This command allows you to manually slip individual sectors on a Xylogics-controlled disk with slip-sectoring enabled. It prompts to see if you want to attempt to save the data in the sector being slipped.

`rhdr` (read headers — Xylogics 450 only). This is among the most informative commands; it displays individual sector headers sequentially. Use it to help identify slipped or mapped sectors, to show whether the disk is configured for sector slipping, and to help spot anomalies in sector headers.

`label` (write label on disk) — This command writes a label on the disk.

`partition` (set partition table) — This test allows you to set the partition table boundaries.

`scan` — This **destructive** command does repeated sector scans over a specified range of the disk. On SMD disks, it can automatically map or slip any bad sectors it finds. Use this command after formatting the disk, but before installing UNIX, to check for bad sectors that didn't show up on other tests.

`whdr` (Xylogics 450 only) — This command reads in existing sector headers starting at a specified location, then loops while asking if you want to change a header. **SPECIAL USES ONLY.**

`format` — `diag` actually provides a SCSI format subsystem with its own `format` subcommand, and an SMD subsystem with its own `format` subcommand. Format subsystems are where you go to perform formatting related operations, and the format subcommands actually cause the format to occur.

format

Formatting a disk describes the process of dividing it into sectors so UNIX can use it. `diag` provides separate format subsystems for both SMD and SCSI disks; these are invoked by entering the command `format` to the `diag` prompt. `diag` selects the proper subsystem based on configuration information.

Once within either format subsystem, the prompt changes to:

```
format>
```

Both format subsystems provide `format` commands; these actually perform the format operation to the disk. For instructions to use the `format` commands, see the instructions earlier in this chapter.

map

CAUTION **This command can destroy disk data if you add a new mapping.** Backup disk data before proceeding.

The `map` command (SMD disks only) displays the current map table and allows you to add a new mapping.

After you enter `map`, it displays the current map table, then prompts for additional information. When it asks if you want to add a mapping, if you enter `n`, it exits without changing anything. If you enter `y` to add a mapping, it asks if you want to attempt to preserve the data. If you answer `y`, it writes the data to the alternate sector before the mapping is actually done. A typical session goes:

```
diag> map
Current mapping:
Sector CC/HH/SS mapped to CC/HH/SS
Sector CC/HH/SS mapped to CC/HH/SS
Do you wish to add a mapping? y
mapping may be removed only by complete format of the disk
cylinder to be mapped? 704
track to be mapped? 9
sector to be mapped? 7
Attempt to preserve data? y
Data transfer successful!
OK to map 704/9/7? y
mapped sector 704/7/7 to 822/8/31
```

After it maps the sector, `map` adds the new sector to the map, marks the old sector bad, and rewrites the new map table.

If a read error occurs during attempt to preserve data, `map` reports the transfer unsuccessful. This may be only partially true; with fixed ECC errors, the data is still intact. With a hard CRC error, some data may survive, but with other errors the data is usually lost.

You can map a sector on a disk set up for slip sectoring, but slipping a bad sector is preferable to mapping it.

fix

CAUTION **This command damages disk data.** Backup disk data before proceeding.

The `fix` command formats and verifies user-specified sections of SMD disks. Use it to verify a section of the disk without doing the entire thing.

`Fix` requires a starting and ending track address. It also requires a number of surface analysis passes. After it obtains this information, it asks for permission to continue.

A typical session might go:

```
diag> fix
fix -- DESTROYS SOME DISK DATA
formats a range of tracks
enter track number as 'cyl/track'
starting track? 15/20
ending track? 15/30
# of surface analysis passes (5 recommended)? 5
OK to format from 15/20/0 to 15/29/31? y
CC/HH
diag>
```

CC/HH represents the current track number. It increments as the test proceeds.

If you answer `y`, it proceeds like a format within the specified boundaries, except that it prints cylinder/head numbers instead of just cylinder numbers. If it encounters a previously slipped sector, it reslips the sector after the track is formatted, and displays a message announcing this fact. It reports mapped sectors and adds them to the map if they are not already there.

Before it exits, it writes the new bad sector table on the disk then updates the mapped sector headings. If it is interrupted, it does not write the new map or update the mapped sector headers.

slip

CAUTION **This command may destroy disk data.** Backup disk data before proceeding.

The slip command allows you to manually slip a sector on an SMD disk connected to a Xylogics 450 controller, provided the disk and the sector are eligible for sector slipping.

`slip` asks if you want to save data; if you answer `y`, it stores the data from the entire track in a buffer, and attempts to rewrite the data after the slip. The data usually survives ECC errors; other errors may cause damage.

rhdr

The command `rhdr` (read headers) works with Xylogics 450 controllers only. It displays the sector headers for consecutive tracks.

A display similar to the following results:

```
diag> rhdr
read track header
starting track (dd/dd): 00

0/0
sec 0  8000  8100  8200  8300  8400  8500  8600  8700
sec 8  8800  8900  8A00  8B00  8C00  8D00  8E00  8F00
sec 16 9000  9100  9200  9300  9400  9500  9600  9700
sec 24 9800  9900  9A00  9B00  9C00  9D00  9E00  9F00
stop? <ret>
0/1
sec 0  9F01  8001  8101  8201  8301  8401  8501  8601
sec 8  8701  8801  8901  8A01  8B01  8C01  8D01  8E01
sec 16 8F01  9001  9101  9201  9301  9401  9501  9601
sec 24 9700  9801  9901  9A01  9B01  9C01  9D01  9E01
stop? y
```

Reading headers can help locate anomalies in sector headers, and locate spare, mapped, slipped and runt sectors. This in turn can help identify a disk with slip sectoring. The following headers identify unusual sectors:

FFFFFFFF — Mapped sector. These headings identify mapped sectors. UNIX redirects accesses to these sectors to spare sectors as specified by the map. On a disk with slip sectoring, mapping occurs only after a second sector on a track goes bad (very rarely).

FEFEFEFE — Slipped sector. This header marks a place where a logical sector was slipped from.

DDDDDDDD — Spare data sector. Each track on a slip sectored disk starts life with a spare data sector; if a bad sector on that track is slipped, this spare gets used up.

EEEEEEEE — Runt sector. This is the designation given to the extra space at the end of a track; it is usually too short for a data sector but too long to ignore.

Use `rhdr` to help identify mapped or slipped sectors, or to discover if a disk is setup for slip sectoring. Also, if any error message identifies a track or sector as the source of a problem, use `rhdr` to check the headers in that area for consistency.

Understanding a normal header number requires translating it from hex to binary and reading the contents of the fields. The bits are:

ss00 0ccc cccc cccc ttss ssss hhhh hhhh

where:

h = head number
s = sector number (note s bits in two locations!)
t = drive type
c = cylinder number
0 = unused bits, always set to 0

The head number is straightforward; it is two hex numbers.

The sector number is divided into two fields, but the two leading 's' bits are (normally) 0s, as almost all disks have fewer than 64 sectors on a track.

The drive type is an identifier for the Xylogics controller. These are hardwired into `diag` except in case of *other* disk types. The usual assignments for these bits are:

00	Fujitsu-M2351 Eagle
01	Fujitsu-M2312K
10	Fujitsu-M2284/2322
11	Other

The cylinder number is also straightforward; three hex numbers where the most significant hex digit is <8 because its high-order bit is 0.

When `diag` displays numbers, it strips off leading 0s. In the above header display, the first header with all zeroes attached would be 00008000.

The following example shows the decoding of the last header number shown in the `rhdr` display above:

9E01 - as shown

00009E01 - append leading zeroes

`0000 0000 0000 0000 1001 1110 0000 0001` - translate to binary

`ss00 0ccc cccc cccc ttss ssss hhhh hhhh` - show field values

This shows it is cylinder 0, head 1, sector 0x1E, on a drive type 2 (10 binary - Fujitsu-M2284/2322).

label

NOTE *If you write an incorrect label on a disk, UNIX may not be able to use a filesystem. However, if you rewrite a correct label it should correct the problem.*

The `label` command writes a new label on the disk. It asks if you want to use the 'built in' (default) partition map for your disk, and displays a copy of what it has done.

A typical session goes:

```
diag> label
label this disk
OK to use logical partition map disk_type? y
Are you sure you want to write? y
```

After it writes the label, it displays the label it has just written. For example, if you had a Fujitsu M2322, it would display:

```
verify label
id: <fujitsu-M2322 cyl 821 alt 2 hd 10 sec 32 interleave 1>
    Partition a: starting cyl=0, # blocks=15884
    Partition b: starting cyl=50, # blocks=33440
    Partition c: starting cyl=0, # blocks=262720
    Partition g: starting cyl=155, # blocks=213120
diag>
```

Note that the numbers in the above display differ depending on disk type.

To use a label other than the 'built in' defaults, see *partition*, and *Installing UNIX on the Sun Workstation*.

partition

The `partition` command selects a table to use when labeling the disk. `Diag` maintains default partition tables for all standard disks; it asks you to select a disk, then prints the default partition table. Then it asks you if you want to modify this table. A typical session goes:

```
diag> partition
Select partition table
   0 - Fujitsu-M2312K
   1 - Fujitsu-M2284/2322
   2 - Fujitsu-M2284/2322 alternate
   3 - Fujitsu-M2351 Eagle
   4 - Fujitsu-M2351 Eagle alternate
   5 - Fujitsu-M2351 Eagle standalone
   6 - Other
Which One? 1
Do you wish to modify this table? y
Partition a: starting cyl=0, # blocks=15884
Change this partition? n
Partition b: starting cyl=50, # blocks=33440
Change this partition? n
Partition c: starting cyl=0, # blocks=262720
Change this partition? n
Partition d: starting cyl=0, # blocks=0
Change this partition? n
Partition e: starting cyl=0, # blocks=0
Change this partition? n
Partition f: starting cyl=0, # blocks=0
Change this partition? n
Partition g: starting cyl=155, # blocks=213120
Change this partition? y
starting cylinder? 155
# of blocks? 600/0/0
Partition h: starting cyl=0, # blocks=0
Change this partition? y
starting cylinder? 755
# of blocks 66/0/0
Verify partition table 'Fujitsu-M2284/2322':
        Partition a: starting cyl=0, # blocks=15884
        Partition b: starting cyl=50, # blocks=33440
        Partition c: starting cyl=0, # blocks=262720
        Partition d: starting cyl=0, # blocks=0
        Partition e: starting cyl=0, # blocks=0
        Partition f: starting cyl=0, # blocks=0
        Partition g: starting cyl=155, # blocks=192000
        Partition h: starting cyl=755, # blocks=21120
OK to use this partition table? y
Use the label command to write out the partition table.
```

Note that if you select 'other', it asks you to name the partition table, and then it goes through the above sequence, except that all values start at 0.

scan

The `scan` command performs repeated sector scans over a specified range of the disk. It is typically used to find additional disk errors after the disk is formated but before UNIX is installed.

CAUTION **This command destroys disk data. Backup disk data before proceeding.**

`scan` looks for new bad sectors and doesn't look at look at mapped sectors. Unlike `fix`, when it does a mapping, it writes the new mapping table to the disk immediately. It runs continuously until interrupted.

`scan` includes a number of options, all of which it prompts for. These are:

scan entire disk? — If you answer `y`, it scans the entire disk; if you answer `n`, it asks for beginning and ending addresses. If the area to scan includes primary and secondary label areas, it displays a message to this effect.

NOTE *If you overwrite the disk label, you will have to re-label the disk before use.*

use random bit patterns? — `y` causes it to use random patterns (better for a small intensive disk scans); `n` causes it to use 5 standard patterns.

perform corrections when defects are found?— If `diag` is configured for a Xylogics controller, `scan` asks if it should do corrections to bad sectors. If `y`, it tries to slip bad sectors and if it can't, it tries to map them. If `n`, it only reports newly found bad sectors. If the controller is not a Xylogics, it displays a message that it cannot fix defective sectors.

Using `scan` without fixing bad sectors is handy for tracking cable problems, or for other cases where you don't want to fix a sector every time you find an error. It is also useful for SCSI surface analysis.

A typical session might go:

```
diag> scan
scan - continuous scan for defective sectors
DESTROYS DISK DATA
scan entire disk? n
starting block? 700
ending block? 702
use random bit patterns? y
perform corrections when errors are found? y
OK to scan from 2/1/25 to 2/1/26? y
type control-C to quit

[pass 1 - bit pattern #1: 0xnnnnnnnn]
2/1/n
```

Scan continues until the user aborts with `control C`. Then it prints:

```
Command aborted
diag>
```

whdr

Caution: This command destroys disk data. Backup disk data before proceeding.

The `whdr` command (Xylogics 450 only) changes sector headers on request. This requires intimate knowledge of sector headers and should only be necessary under extreme circumstances.

A

File System Check Program

File System Check Program **259**

A.1. Overview of the File System 259
Superblock 259
Summary Information 260
Cylinder Groups 260
Fragments 261
Updates to the File System 261
A.2. Fixing Corrupted File Systems 262
Detecting and Correcting Corruption 262
Super-block Checking 263
Free Block Checking 263
Checking the Inode State 263
Inode Links 264
Inode Data Size 264
Checking the Data Associated with an Inode 265
File System Connectivity 265
A.3. Fsck Error Conditions 265
Conventions 265
Initialization 266
Phase 1 Check Blocks and Sizes 268
Phase 1B: Rescan for More Dup's 271
Phase 2 Check Pathnames 271
Phase 3 Check Connectivity 273

Phase 4 Check Reference Counts 274
Phase 5 - Check Cyl Groups 277
Phase 6 - Salvage Cylinder Groups 279
Cleanup 279
A.4. References 279

A

File System Check Program

For Release 3.0 the file system blocksize has been changed to 8192. This document has not yet been updated to reflect that change.

When a UNIX† operating system is brought up, a consistency check of the file systems should always be performed. This precautionary measure helps to insure a reliable environment for file storage on disk. If an inconsistency is discovered, corrective action must be taken. `fsck` runs in two modes. Normally it is run non-interactively by the system after a normal boot. When running in this mode, it will only make changes to the file system that are known to always be correct. If an unexpected inconsistency is found `fsck` will exit with a non-zero exit status, leaving the system running single-user. Typically the operator then runs `fsck` interactively. When running in this mode, each problem is listed followed by a suggested corrective action. The operator must decide whether or not the suggested correction should be made.

The purpose of this memo is to dispel the mystique surrounding file system inconsistencies. It first describes the updating of the file system (the calm before the storm) and then describes file system corruption (the storm). Finally, the set of deterministic corrective actions used by `fsck` (the Coast Guard to the rescue) is presented.

A.1. Overview of the File System

The file system is discussed in detail in [Mckusick83]; this section gives a brief overview.

Superblock

A file system is described by its *super-block*. The super-block is built when the file system is created (see *newfs* (8)) and never changes. The super-block contains the basic parameters of the file system, such as the number of data blocks it contains and a count of the maximum number of files. Because the super-block contains critical data, *newfs* replicates it to protect against catastrophic loss. The *default super block* always resides at a fixed offset from the beginning of the file system's disk partition. The *redundant super blocks* are not referenced unless a head crash or other hard disk error causes the default super-block to be unusable. The redundant blocks are sprinkled throughout the disk partition.

This document reflects the use of `fsck` with the revised file system organization implemented in release 0.1 of the Sun UNIX operating system. This is a revision of the original paper written by T. J. Kowalski.

† UNIX is a trademark of AT&T Bell Laboratories.

Within the file system are files. Certain files are distinguished as directories and contain collections of pointers to files that may themselves be directories. Every file has a descriptor associated with it called an *inode*. The inode contains information describing ownership of the file, time stamps indicating modification and access times for the file, and an array of indices pointing to the data blocks for the file. In this section, we assume that the first 12 blocks of the file are directly referenced by values stored in the inode structure itself†. The inode structure may also contain references to indirect blocks containing further data block indices. In a file system with a 4096 byte block size, a singly indirect block contains 1024 further block addresses, a doubly indirect block contains 1024 addresses of further single indirect blocks, and a triply indirect block contains 1024 addresses of further doubly indirect blocks.

Sun's block size is 4K; fragment size is 1K.

In order to create files with up to $2 sup 32$ bytes, using only two levels of indirection, the minimum size of a file system block is 4096 bytes. The size of file system blocks can be any power of two greater than or equal to 4096. The block size of the file system is maintained in the super-block, so it is possible for file systems of different block sizes to be accessible simultaneously on the same system. The block size must be decided when *newfs* creates the file system; the block size cannot be subsequently changed without rebuilding the file system.

Summary Information

Associated with the super block is non replicated *summary information*. The summary information changes as the file system is modified. The summary information contains the number of blocks, fragments, inodes and directories in the file system.

Cylinder Groups

The file system partitions the disk into one or more areas called *cylinder groups*. A cylinder group is comprised of one or more consecutive cylinders on a disk. Each cylinder group includes inode slots for files, a *block map* describing available blocks in the cylinder group, and summary information describing the usage of data blocks within the cylinder group. A fixed number of inodes is allocated for each cylinder group when the file system is created. The current policy is to allocate one inode for each 2048 bytes of disk space; this is expected to be far more inodes than will ever be needed.

All the cylinder group bookkeeping information could be placed at the beginning of each cylinder group. However if this approach were used, all the redundant information would be on the top platter. A single hardware failure that destroyed the top platter could cause the loss of all copies of the redundant super-blocks. Thus the cylinder group bookkeeping information begins at a floating offset from the beginning of the cylinder group. The offset for the $i+1$ st cylinder group is about one track further from the beginning of the cylinder group than it was for the i th cylinder group. In this way, the redundant information spirals down into the pack; any single track, cylinder, or platter can be lost without losing all copies of the super-blocks. Except for the first cylinder group, the space between the beginning of the cylinder group and the beginning of the cylinder group

†The actual number may vary from system to system, but is usually in the range 5-13.

information stores data.

Fragments

To avoid waste in storing small files, the file system space allocator divides a single file system block into one or more *fragments*. The fragmentation of the file system is specified when the file system is created; each file system block can be optionally broken into 2, 4, or 8 addressable fragments. The lower bound on the size of these fragments is constrained by the disk sector size; typically 512 bytes is the lower bound on fragment size. The block map associated with each cylinder group records the space availability at the fragment level. Aligned fragments are examined to determine block availability.

On a file system with a block size of 4096 bytes and a fragment size of 1024 bytes, a file is represented by zero or more 4096 byte blocks of data, and possibly a single fragmented block. If a file system block must be fragmented to obtain space for a small amount of data, the remainder of the block is made available for allocation to other files. For example, consider an 11000 byte file stored on a 4096/1024 byte file system. This file uses two full size blocks and a 3072 byte fragment. If no fragments with at least 3072 bytes are available when the file is created, a full size block is split yielding the necessary 3072 byte fragment and an unused 1024 byte fragment. This remaining fragment can be allocated to another file, as needed.

Updates to the File System

Every working day hundreds of files are created, modified, and removed. Every time a file is modified, the operating system performs a series of file system updates. These updates, when written on disk, yield a consistent file system. The file system stages all modifications of critical information; modification can either be completed or cleanly backed out after a crash. Knowing the information that is first written to the file system, deterministic procedures can be developed to repair a corrupted file system. To understand this process, the order that the update requests were being honored must first be understood.

When a user program does an operation to change the file system, such as a *write*, the data to be written is copied into an internal *in-core* buffer in the kernel. Normally, the disk update is handled asynchronously; the user process is allowed to proceed even though the data has not yet been written to the disk. The data, along with the inode information reflecting the change, is eventually written out to disk. The real disk write may not happen until long after the *write* system call has returned. Thus at any given time, the file system, as it resides on the disk, lags behind the state of the file system represented by the in-core information.

The disk information is updated to reflect the in-core information when the buffer is required for another use, when a *sync* (2) is done (at 30 second intervals) by */etc/update* (8), or by manual operator intervention with the *sync* (8) command. If the system is halted without writing out the in-core information, the file system on the disk will be in an inconsistent state.

If all updates are done asynchronously, several serious inconsistencies can arise. One inconsistency is that a block may be claimed by two inodes. Such an inconsistency can occur when the system is halted before the pointer to the block in the old inode has been cleared in the copy of the old inode on the disk, and after

the pointer to the block in the new inode has been written out to the copy of the new inode on the disk. Here, there is no deterministic method for deciding which inode should really claim the block. A similar problem can arise with a multiply claimed inode.

The problem with asynchronous inode updates can be avoided by doing all inode deallocations synchronously. Consequently, inodes and indirect blocks are written to the disk synchronously (*i.e.* the process blocks until the information is really written to disk) when they are being deallocated. Similarly inodes are kept consistent by synchronously deleting, adding, or changing directory entries.

A.2. Fixing Corrupted File Systems

A file system can become corrupted in several ways. The most common of these ways are improper shutdown procedures and hardware failures.

File systems may become corrupted during an *unclean halt*. This happens when proper shutdown procedures are not observed, physically write-protecting a mounted file system, or a mounted file system is taken off-line. The most common operator procedural failure is forgetting to *sync* the system before halting the CPU.

File systems may become further corrupted if proper startup procedures are not observed, e.g., not checking a file system for inconsistencies, and not repairing inconsistencies. Allowing a corrupted file system to be used (and, thus, to be modified further) can be disastrous.

Any piece of hardware can fail at any time. Failures can be as subtle as a bad block on a disk pack, or as blatant as a non-functional disk-controller.

Detecting and Correcting Corruption

Normally `fsck` is run non-interactively. In this mode it will only fix corruptions that are expected to occur from an unclean halt. These actions are a proper subset of the actions that `fsck` will take when it is running interactively. Throughout this paper we assume that `fsck` is being run interactively, and all possible errors can be encountered. When an inconsistency is discovered in this mode, `fsck` reports the inconsistency for the operator to chose a corrective action.

A quiescent‡ file system may be checked for structural integrity by performing consistency checks on the redundant data intrinsic to a file system. The redundant data is either read from the file system, or computed from other known values. The file system must be in a quiescent state when `fsck` is run, since `fsck` is a multi-pass program.

In the following sections, we discuss methods to discover inconsistencies and possible corrective actions for the cylinder group blocks, the inodes, the indirect blocks, and the data blocks containing directory entries.

‡ I.e., unmounted and not being written on.

Super-block Checking

The most commonly corrupted item in a file system is the summary information associated with the super-block. The summary information is prone to corruption because it is modified with every change to the file system's blocks or inodes, and is usually corrupted after an unclean halt.

The super-block is checked for inconsistencies involving file-system size, number of inodes, free-block count, and the free-inode count. The file-system size must be larger than the number of blocks used by the super-block and the number of blocks used by the list of inodes. The file-system size and layout information are the most critical pieces of information for `fsck`. While there is no way to actually check these sizes, since they are statically determined by *newfs*, `fsck` can check that these sizes are within reasonable bounds. All other file system checks require that these sizes be correct. If `fsck` detects corruption in the static parameters of the default super-block, `fsck` requests the operator to specify the location of an alternate super-block.

Free Block Checking

`fsck` checks that all the blocks marked as free in the cylinder group block maps are not claimed by any files. When all the blocks have been initially accounted for, `fsck` checks that the number of free blocks plus the number of blocks claimed by the inodes equals the total number of blocks in the file system.

If anything is wrong with the block allocation maps, `fsck` will rebuild them, based on the list it has computed of allocated blocks.

The summary information associated with the super-block counts the total number of free blocks within the file system. `fsck` compares this count to the number of free blocks it found within the file system. If the two counts do not agree, then `fsck` replaces the incorrect count in the summary information by the actual free-block count.

The summary information counts the total number of free inodes within the file system. `fsck` compares this count to the number of free inodes it found within the file system. If the two counts do not agree, then `fsck` replaces the incorrect count in the summary information by the actual free-inode count.

Checking the Inode State

An individual inode is not as likely to be corrupted as the allocation information. However, because of the great number of active inodes, a few of the inodes are usually corrupted.

The list of inodes in the file system is checked sequentially starting with inode 2 (inode 0 marks unused inodes; inode 1 is saved for future generations) and progressing through the last inode in the file system. The state of each inode is checked for inconsistencies involving format and type, link count, duplicate blocks, bad blocks, and inode size.

Each inode contains a mode word. This mode word describes the type and state of the inode. Inodes must be one of six types: regular inode, directory inode, symbolic link inode, special block inode, special character inode, or socket inode. Inodes may be found in one of three allocation states: unallocated, allocated, and neither unallocated nor allocated. This last state suggests an incorrectly formated inode. An inode can get in this state if bad data is written into the inode list. The only possible corrective action is for `fsck` is to clear the

inode.

Inode Links

Each inode counts the total number of directory entries linked to the inode. `fsck` verifies the link count of each inode by starting at the root of the file system, and descending through the directory structure. The actual link count for each inode is calculated during the descent.

If the stored link count is non-zero and the actual link count is zero, then no directory entry appears for the inode. If this happens, `fsck` will place the disconnected file in the *lost+found* directory. If the stored and actual link counts are non-zero and unequal, a directory entry may have been added or removed without the inode being updated. If this happens, `fsck` replaces the incorrect stored link count by the actual link count.

Each inode contains a list, or pointers to lists (indirect blocks), of all the blocks claimed by the inode. Since indirect blocks are owned by an inode, inconsistencies in indirect blocks directly affect the inode that owns it.

`fsck` compares each block number claimed by an inode against a list of already allocated blocks. If another inode already claims a block number, then the block number is added to a list of *duplicate blocks*. Otherwise, the list of allocated blocks is updated to include the block number.

If there are any duplicate blocks, `fsck` will perform a partial second pass over the inode list to find the inode of the duplicated block. The second pass is needed, since without examining the files associated with these inodes for correct content, not enough information is available to determine which inode is corrupted and should be cleared. If this condition does arise (only hardware failure will cause it), then the inode with the earliest modify time is usually incorrect, and should be cleared. If this happens, `fsck` prompts the operator to clear both inodes. The operator must decide which one should be kept and which one should be cleared.

`fsck` checks the range of each block number claimed by an inode. If the block number is lower than the first data block in the file system, or greater than the last data block, then the block number is a *bad block number*. Many bad blocks in an inode are usually caused by an indirect block that was not written to the file system, a condition which can only occur if there has been a hardware failure. If an inode contains bad block numbers, `fsck` prompts the operator to clear it.

Inode Data Size

Each inode contains a count of the number of data blocks that it contains. The number of actual data blocks is the sum of the allocated data blocks and the indirect blocks. `fsck` computes the actual number of data blocks and compares that block count against the actual number of blocks the inode claims. If an inode contains an incorrect count `fsck` prompts the operator to fix it.

Each inode contains a thirty-two bit size field. The size is the number of data bytes in the file associated with the inode. The consistency of the byte size field is roughly checked by computing from the size field the maximum number of blocks that should be associated with the inode, and comparing that expected block count against the actual number of blocks the inode claims.

Checking the Data Associated with an Inode

An inode can directly or indirectly reference three kinds of data blocks. All referenced blocks must be the same kind. The three types of data blocks are: plain data blocks, symbolic link data blocks, and directory data blocks. Plain data blocks contain the information stored in a file; symbolic link data blocks contain the path name stored in a link. Directory data blocks contain directory entries. `fsck` can only check the validity of directory data blocks.

Each directory data block is checked for several types of inconsistencies. These inconsistencies include directory inode numbers pointing to unallocated inodes, directory inode numbers that are greater than the number of inodes in the file system, incorrect directory inode numbers for "." and "..", and directories that are not attached to the file system. If the inode number in a directory data block references an unallocated inode, then `fsck` will remove that directory entry. Again, this condition can only arise when there has been a hardware failure.

If a directory entry inode number references outside the inode list, then `fsck` will remove that directory entry. This condition occurs if bad data is written into a directory data block.

The directory inode number entry for "." must be the first entry in the directory data block. The inode number for "." must reference itself; e.g., it must equal the inode number for the directory data block. The directory inode number entry for ".." must be the second entry in the directory data block. Its value must equal the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory). If the directory inode numbers are incorrect, `fsck` will replace them with the correct values.

File System Connectivity

`fsck` checks the general connectivity of the file system. If directories are not linked into the file system, then `fsck` links the directory back into the file system in the *lost+found* directory. This condition only occurs when there has been a hardware failure.

A.3. Fsck Error Conditions Conventions

`fsck` is a multi-pass file system check program. Each file system pass invokes a different Phase of the `fsck` program. After the initial setup, `fsck` performs successive Phases over each file system, checking blocks and sizes, path-names, connectivity, reference counts, and the map of free blocks, (possibly rebuilding it), and performs some cleanup.

Normally `fsck` is run non-interactively to *preen* the file systems after an unclean halt. While preening a file system, it will only fix corruptions that are expected to occur from an unclean halt. These actions are a proper subset of the actions that `fsck` will take when it is running interactively. Throughout this appendix many errors have several options that the operator can take. When an inconsistency is detected, `fsck` reports the error condition to the operator. If a response is required, `fsck` prints a prompt message and waits for a response. When preen'ing most errors are fatal. For those that are expected, the response taken is noted. This appendix explains the meaning of each error condition, the possible responses, and the related error conditions.

The error conditions are organized by the *Phase* of the `fsck` program in which they can occur. The error conditions that may occur in more than one Phase will be discussed in initialization.

Initialization

Before a file system check can be performed, certain tables have to be set up and certain files opened. This section concerns itself with the opening of files and the initialization of tables. This section lists error conditions resulting from command line options, memory requests, opening of files, status of files, file system size checks, and creation of the scratch file. All of the initialization errors are fatal when the file system is being preen'ed.

_C_ option?
C is not a legal option to `fsck`; legal options are –b, –y, –n, and –p. `fsck` terminates on this error condition. See the `fsck`(8) manual entry for further detail.

cannot alloc NNN bytes for blockmap
cannot alloc NNN bytes for freemap
cannot alloc NNN bytes for statemap
cannot alloc NNN bytes for lncntp
`fsck`'s request for memory for its virtual memory tables failed. This should never happen. `fsck` terminates on this error condition. See a guru.

Can't open checklist file: *F*
The file system checklist file *F* (usually */etc/fstab*) can not be opened for reading. `fsck` terminates on this error condition. Check access modes of *F*.

Can't stat root
`fsck`'s request for statistics about the root directory "/" failed. This should never happen. `fsck` terminates on this error condition. See a guru.

Can't stat *F*
Can't make sense out of name *F*
`fsck`'s request for statistics about the file system *F* failed. When running manually, it ignores this file system and continues checking the next file system given. Check access modes of *F*.

Can't open *F*
`fsck`'s request attempt to open the file system *F* failed. When running manually, it ignores this file system and continues checking the next file system given. Check access modes of *F*.

***F*: (NO WRITE)**
Either the –n flag was specified or `fsck`'s attempt to open the file system *F* for writing failed. When running manually, all the diagnostics are printed out, but no modifications are attempted to fix them.

file is not a block or character device; OK
You have given `fsck` a regular file name by mistake. Check the type of the file specified.

Possible responses to the OK prompt are:

YES
Ignore this error condition.

NO
ignore this file system and continues checking the next file system given.

One of the following messages will appear:
MAGIC NUMBER WRONG
NCG OUT OF RANGE
CPG OUT OF RANGE
NSECT < 1
NTRAK < 1
SPC DOES NOT JIVE w/NTRAK*NSECT
INODES NOT MULTIPLE OF A BLOCK
IMPLIES MORE INODE THAN DATA BLOCKS
NCYL DOES NOT JIVE WITH NCG*CPG
FPG DOES NOT JIVE WITH CPG & SPC
SIZE PREPOSTEROUSLY SMALL
SIZE PREPOSTEROUSLY LARGE
CGSIZE INCORRECT
CSSIZE INCORRECT

and will be followed by the message:
F*: BAD SUPER BLOCK: *B
USE -b OPTION TO FSCK TO SPECIFY LOCATION OF AN ALTERNATE
SUPER-BLOCK TO SUPPLY NEEDED INFORMATION; SEE fsck(8).
The super block has been corrupted. An alternative super block must be selected from among those listed by *newfs* (8) when the file system was created. For file systems with a blocksize less than 32K, specifying –b32 is a good first choice.

CAN NOT SEEK: BLK *B* (CONTINUE)
`fsck`'s request for moving to a specified block number *B* in the file system failed. This should never happen. See a guru.

Possible responses to the CONTINUE prompt are:

YES
attempt to continue to run the file system check. Often, however the problem will persist. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to re-check this file system. If the block was part of the virtual memory buffer cache, `fsck` will terminate with the message ''Fatal I/O error''.

NO
terminate the program.

CAN NOT READ: BLK *B* (CONTINUE)
`fsck`'s request for reading a specified block number *B* in the file system failed. This should never happen. See a guru.

Possible responses to the CONTINUE prompt are:

YES
attempt to continue to run the file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to re-check this file system. If the block was part of the virtual memory buffer cache, `fsck` will terminate with the message ''Fatal I/O error''.

NO
terminate the program.

CAN NOT WRITE: BLK *B* (CONTINUE)
`fsck`'s request for writing a specified block number *B* in the file system failed. The disk is write-protected. See a guru.

Possible responses to the CONTINUE prompt are:

YES
attempt to continue to run the file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to re-check this file system. If the block was part of the virtual memory buffer cache, `fsck` will terminate with the message ''Fatal I/O error''.

NO
terminate the program.

Phase 1 – Check Blocks and Sizes

This phase concerns itself with the inode list. This section lists error conditions resulting from checking inode types, setting up the zero-link-count table, examining inode block numbers for bad or duplicate blocks, checking inode size, and checking inode format. All errors in this phase except **INCORRECT BLOCK COUNT** are fatal if the file system is being preen'ed,

cg *C*: bad magic number The magic number of cylinder group *C* is wrong. This usually indicates that the cylinder group maps have been destroyed. When running manually the cylinder group is marked as needing to be reconstructed.

UNKNOWN FILE TYPE I=*I* (CLEAR) The mode word of the inode *I* indicates that the inode is not a special block inode, special character inode, socket inode, regular inode, symbolic link, or directory inode.

Possible responses to the CLEAR prompt are:

YES
de-allocate inode *I* by zeroing its contents. This will always invoke the UNALLOCATED error condition in Phase 2 for each directory entry pointing to this inode.

NO
ignore this error condition.

LINK COUNT TABLE OVERFLOW (CONTINUE)
An internal table for `fsck` containing allocated inodes with a link count of zero has no more room. Recompile `fsck` with a larger value of MAXLNCNT.

Possible responses to the CONTINUE prompt are:

YES
continue with the program. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to re-check this file system. If another allocated inode with a zero link count is found, this error condition is repeated.

NO
terminate the program.

B* BAD I=*I
Inode *I* contains block number *B* with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system. This error condition may invoke the **EXCESSIVE BAD BLKS** error condition in Phase 1 if inode *I* has too many block numbers outside the file system range. This error condition will always invoke the **BAD/DUP** error condition in Phase 2 and Phase 4.

EXCESSIVE BAD BLKS I=*I* (CONTINUE)
There is more than a tolerable number (usually 10) of blocks with a number lower than the number of the first data block in the file system or greater than the number of last block in the file system associated with inode *I*.

Possible responses to the CONTINUE prompt are:

YES
ignore the rest of the blocks in this inode and continue checking with the next inode in the file system. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to re-check this file system.

NO
terminate the program.

B* DUP I=*I
Inode *I* contains block number *B* which is already claimed by another inode.

This error condition may invoke the **EXCESSIVE DUP BLKS** error condition in Phase 1 if inode *I* has too many block numbers claimed by other inodes. This error condition will always invoke Phase 1b and the **BAD/DUP** error condition in Phase 2 and Phase 4.

EXCESSIVE DUP BLKS I=*I* (CONTINUE)
There is more than a tolerable number (usually 10) of blocks claimed by other inodes.

Possible responses to the CONTINUE prompt are:

YES
: ignore the rest of the blocks in this inode and continue checking with the next inode in the file system. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to re-check this file system.

NO
: terminate the program.

DUP TABLE OVERFLOW (CONTINUE)
An internal table in `fsck` containing duplicate block numbers has no more room. Recompile `fsck` with a larger value of DUPTBLSIZE.

Possible responses to the CONTINUE prompt are:

YES
: continue with the program. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to re-check this file system. If another duplicate block is found, this error condition will repeat.

NO
: terminate the program.

PARTIALLY ALLOCATED INODE I=*I* (CLEAR)
Inode *I* is neither allocated nor unallocated.

Possible responses to the CLEAR prompt are:

YES
: de-allocate inode *I* by zeroing its contents.

NO
: ignore this error condition.

INCORRECT BLOCK COUNT I=*I* (*X* should be *Y*) (CORRECT)
The block count for inode *I* is *X* blocks, but should be *Y* blocks. When preen'ing the count is corrected.

Possible responses to the CORRECT prompt are:

YES
replace the block count of inode *I* with *Y*.

NO
ignore this error condition.

Phase 1B: Rescan for More Dup's

When a duplicate block is found in the file system, the file system is rescanned to find the inode which previously claimed that block. This section lists the error condition when the duplicate block is found.

B **DUP I=***I*
Inode *I* contains block number *B* that is already claimed by another inode. This error condition will always invoke the **BAD/DUP** error condition in Phase 2. You can determine which inodes have overlapping blocks by examining this error condition and the DUP error condition in Phase 1.

Phase 2 – Check Pathnames

This phase concerns itself with removing directory entries pointing to error conditioned inodes from Phase 1 and Phase 1b. This section lists error conditions resulting from root inode mode and status, directory inode pointers in range, and directory entries pointing to bad inodes. All errors in this phase are fatal if the file system is being preen'ed.

ROOT INODE UNALLOCATED. TERMINATING.
The root inode (usually inode number 2) has no allocate mode bits. This should never happen. The program will terminate.

NAME TOO LONG *F*
An excessively long path name has been found. This is usually indicative of loops in the file system name space. This can occur if the super user has made circular links to directories. The offending links must be removed (by a guru).

ROOT INODE NOT DIRECTORY (FIX)
The root inode (usually inode number 2) is not directory inode type.

Possible responses to the FIX prompt are:

YES
replace the root inode's type to be a directory. If the root inode's data blocks are not directory blocks, a VERY large number of error conditions will be produced.

NO
terminate the program.

DUPS/BAD IN ROOT INODE (CONTINUE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks in the root inode (usually inode number 2) for the file system.

Possible responses to the CONTINUE prompt are:

YES
ignore the **DUPS/BAD** error condition in the root inode and attempt to continue to run the file system check. If the root inode is not correct, then this may result in a large number of other error conditions.

NO
terminate the program.

I OUT OF RANGE I=*I* NAME=*F* (REMOVE)
A directory entry *F* has an inode number *I* which is greater than the end of the inode list.

Possible responses to the REMOVE prompt are:

YES
the directory entry *F* is removed.

NO
ignore this error condition.

UNALLOCATED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (REMOVE)
A directory entry *F* has a directory inode *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

YES
the directory entry *F* is removed.

NO
ignore this error condition.

UNALLOCATED I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* FILE=*F* (REMOVE)
A directory entry *F* has an inode *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed.

Possible responses to the REMOVE prompt are:

YES
the directory entry *F* is removed.

NO
ignore this error condition.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* DIR=*F* (REMOVE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, directory inode *I*. The owner *O*, mode *M*, size *S*, modify time

T, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

YES
the directory entry *F* is removed.

NO
ignore this error condition.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* FILE=*F* (REMOVE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed.

Possible responses to the REMOVE prompt are:

YES
the directory entry *F* is removed.

NO
ignore this error condition.

Phase 3 – Check Connectivity

This phase concerns itself with the directory connectivity seen in Phase 2. This section lists error conditions resulting from unreferenced directories, and missing or full *lost+found* directories.

DIRECTORY *D* CORRUPTED (SALVAGE)
A directory with an inconsistent internal state has been found. This error is fatal if the file system is being preen'ed.

Possible responses to the SALVAGE prompt are:

YES
throw away all entries up to the next 512-byte boundary. This rather drastic action can throw away up to 42 entries, and should be taken only after other recovery efforts have failed.

NO
Skip up to the next 512-byte boundary and resume reading, but do not modify the directory.

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT)
The directory inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. When preen'ing, the directory is reconnected if its size is non-zero, otherwise it is cleared.

Possible responses to the RECONNECT prompt are:

YES

reconnect directory inode *I* to the file system in the directory for lost files (usually *lost+found*). This may invoke the *lost+found* error condition in Phase 3 if there are problems connecting directory inode *I* to *lost+found*. This may also invoke the CONNECTED error condition in Phase 3 if the link was successful.

NO

ignore this error condition. This will always invoke the UNREF error condition in Phase 4.

SORRY. NO lost+found DIRECTORY

There is no *lost+found* directory in the root directory of the file system; `fsck` ignores the request to link a directory in *lost+found*. This will always invoke the UNREF error condition in Phase 4. Check access modes of *lost+found*. See `fsck`(8) manual entry for further detail. This error is fatal if the file system is being preen'ed.

SORRY. NO SPACE IN lost+found DIRECTORY

There is no space to add another entry to the *lost+found* directory in the root directory of the file system; `fsck` ignores the request to link a directory in *lost+found*. This will always invoke the UNREF error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make *lost+found* larger. See `fsck`(8) manual entry for further detail. This error is fatal if the file system is being preen'ed.

DIR I=*I1* CONNECTED. PARENT WAS I=*I2*

This is an advisory message indicating a directory inode *I1* was successfully connected to the *lost+found* directory. The parent inode *I2* of the directory inode *I1* is replaced by the inode number of the *lost+found* directory.

Phase 4 – Check Reference Counts

This phase concerns itself with the link count information seen in Phase 2 and Phase 3. This section lists error conditions resulting from unreferenced files, missing or full *lost+found* directory, incorrect link counts for files, directories, symbolic links, or special files, unreferenced files, symbolic links, and directories, bad and duplicate blocks in files, symbolic links, and directories, and incorrect total free-inode counts. All errors in this phase are correctable if the file system is being preen'ed except running out of space in the lost+found directory.

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT)

Inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. When preen'ing the file is cleared if either its size or its link count is zero, otherwise it is reconnected.

Possible responses to the RECONNECT prompt are:

YES
reconnect inode *I* to the file system in the directory for lost files (usually *lost+found*). This may invoke the *lost+found* error condition in Phase 4 if there are problems connecting inode *I* to *lost+found.*

NO
ignore this error condition. This will always invoke the CLEAR error condition in Phase 4.

(CLEAR)
The inode mentioned in the immediately previous error condition can not be reconnected. This cannot occur if the file system is being preen'ed, since lack of space to reconnect files is a fatal error.

Possible responses to the CLEAR prompt are:

YES
de-allocate the inode mentioned in the immediately previous error condition by zeroing its contents.

NO
ignore this error condition.

SORRY. NO lost+found DIRECTORY
There is no *lost+found* directory in the root directory of the file system; `fsck` ignores the request to link a file in *lost+found.* This will always invoke the CLEAR error condition in Phase 4. Check access modes of *lost+found*. This error is fatal if the file system is being preen'ed.

SORRY. NO SPACE IN lost+found DIRECTORY
There is no space to add another entry to the *lost+found* directory in the root directory of the file system; `fsck` ignores the request to link a file in *lost+found.* This will always invoke the CLEAR error condition in Phase 4. Check size and contents of *lost+found.* This error is fatal if the file system is being preen'ed.

LINK COUNT FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)
The link count for inode *I* which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

YES
replace the link count of file inode *I* with *Y*.

NO
ignore this error condition.

LINK COUNT DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)
The link count for inode *I* which is a directory, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

YES
replace the link count of directory inode *I* with *Y*.

NO
ignore this error condition.

LINK COUNT *F* I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)
The link count for *F* inode *I* is *X* but should be *Y*. The name *F*, owner *O*, mode *M*, size *S*, and modify time *T* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

YES
replace the link count of inode *I* with *Y*.

NO
ignore this error condition.

UNREF FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)
Inode *I* which is a file, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. When preen'ing, this is a file that was not connected because its size or link count was zero, hence it is cleared.

Possible responses to the CLEAR prompt are:

YES
de-allocate inode *I* by zeroing its contents.

NO
ignore this error condition.

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)
Inode *I* which is a directory, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. When preen'ing, this is a directory that was not connected because its size or link count was zero, hence it is cleared.

Possible responses to the CLEAR prompt are:

YES
de-allocate inode *I* by zeroing its contents.

NO
ignore this error condition.

BAD/DUP FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with file inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. This error cannot arise when the file system is being preen'ed, as it would have caused a fatal error earlier.

Possible responses to the CLEAR prompt are:

YES
de-allocate inode *I* by zeroing its contents.

NO
ignore this error condition.

BAD/DUP DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. This error cannot arise when the file system is being preen'ed, as it would have caused a fatal error earlier.

Possible responses to the CLEAR prompt are:

YES
de-allocate inode *I* by zeroing its contents.

NO
ignore this error condition.

FREE INODE COUNT WRONG IN SUPERBLK (FIX)
The actual count of the free inodes does not match the count in the super-block of the file system. When preen'ing, the count is fixed.

Possible responses to the FIX prompt are:

YES
replace the count in the super-block by the actual count.

NO
ignore this error condition.

Phase 5 - Check Cyl Groups

This phase concerns itself with the free-block maps. This section lists error conditions resulting from allocated blocks in the free-block maps, free blocks missing from free-block maps, and the total free-block count incorrect.

cg *C*: bad magic number
The magic number of cylinder group *C* is wrong. This usually indicates that the cylinder group maps have been destroyed. When running manually the cylinder group is marked as needing to be reconstructed. This error is fatal if the file

system is being preen'ed.

EXCESSIVE BAD BLKS IN BIT MAPS (CONTINUE)
An inode contains more than a tolerable number (usually 10) of blocks claimed by other inodes or that are out of the legal range for the file system. This error is fatal if the file system is being preen'ed.

Possible responses to the CONTINUE prompt are:

YES
ignore the rest of the free-block maps and continue the execution of `fsck`.

NO
terminate the program.

SUMMARY INFORMATION *T* BAD
where *T* is one or more of:
(INODE FREE)
(BLOCK OFFSETS)
(FRAG SUMMARIES)
(SUPER BLOCK SUMMARIES)
The indicated summary information was found to be incorrect. This error condition will always invoke the **BAD CYLINDER GROUPS** condition in Phase 6. When preen'ing, the summary information is recomputed.

***X* BLK(S) MISSING**
X blocks unused by the file system were not found in the free-block maps. This error condition will always invoke the **BAD CYLINDER GROUPS** condition in Phase 6. When preen'ing, the block maps are rebuilt.

BAD CYLINDER GROUPS (SALVAGE)
Phase 5 has found bad blocks in the free-block maps, duplicate blocks in the free-block maps, or blocks missing from the file system. When preen'ing, the cylinder groups are reconstructed.

Possible responses to the SALVAGE prompt are:

YES
replace the actual free-block maps with a new free-block maps.

NO
ignore this error condition.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)
The actual count of free blocks does not match the count in the super-block of the file system. When preen'ing, the counts are fixed.

Possible responses to the FIX prompt are:

YES
replace the count in the super-block by the actual count.

NO
ignore this error condition.

Phase 6 - Salvage Cylinder Groups

This phase concerns itself with the free-block maps reconstruction. No error messages are produced.

Cleanup

Once a file system has been checked, a few cleanup functions are performed. This section lists advisory messages about the file system and modify status of the file system.

***V* files, *W* used, *X* free (*Y* frags, *Z* blocks)**
This is an advisory message indicating that the file system checked contained *V* files using *W* fragment sized blocks leaving *X* fragment sized blocks free in the file system. The numbers in parenthesis breaks the free count down into *Y* free fragments and *Z* free full sized blocks.

******* REBOOT UNIX *******
This is an advisory message indicating that the root file system has been modified by `fsck`. If UNIX is not rebooted immediately, the work done by `fsck` may be undone by the in-core copies of tables UNIX keeps. When preen'ing, `fsck` will exit with a code of 4. The auto-reboot script interprets an exit code of 4 by issuing a reboot system call.

******* FILE SYSTEM WAS MODIFIED *******
This is an advisory message indicating that the current file system was modified by `fsck`. If this file system is mounted or is the current root file system, `fsck` should be halted and UNIX rebooted. If UNIX is not rebooted immediately, the work done by `fsck` may be undone by the in-core copies of tables UNIX keeps.

A.4. References

[Dolotta78] Dolotta, T. A., and Olsson, S. B. eds., *UNIX User's Manual, Edition 1.1* (January 1978).

[Joy83] Joy, W., Cooper, E., Fabry, R., Leffler, S., McKusick, M., and Mosher, D. *System Interface Overview*, University of California at Berkeley, Computer Systems Research Group Technical Report #4, 1982.

[McKusick83] McKusick, M., Joy, W., Leffler, S., and Fabry, R. *A Fast File System for UNIX*, University of California at Berkeley, Computer Systems Research Group Technical Report #7, 1982.

[Ritchie78] Ritchie, D. M., and Thompson, K., The UNIX Time-Sharing System, *The Bell System Technical Journal* 57,

6 (July-August 1978, Part 2), pp. 1905-29.

[Thompson78] Thompson, K., UNIX Implementation, *The Bell System Technical Journal* **57**, 6 (July-August 1978, Part 2), pp. 1931-46.

B

UNIX File System

UNIX File System 283

B.1. Old File System 284
B.2. New file system organization 285
Optimizing storage utilization 286
File system parameterization 289
Layout policies 290
B.3. Performance 292
B.4. File system functional enhancements 295
Long file names 295
File locking 295
Symbolic links 297
Rename 297
Quotas 297
B.5. Software engineering 298
B.6. Acknowledgements 299
B.7. References 299

B

UNIX File System

For release 3.0 the file system blocksize has been changed to 8192. This document has not yet been updated to reflect that change.

This document describes a reimplementation of the UNIX file system. The reimplementation provides substantially higher throughput rates by using more flexible allocation policies, that allow better locality of reference and that can be adapted to a wide range of peripheral and processor characteristics. The new file system clusters data that is sequentially accessed and provides two block sizes to allow fast access for large files while not wasting large amounts of space for small files. File access rates of up to ten times faster than the traditional UNIX file system are experienced. Long needed enhancements to the user interface are discussed. These include a mechanism to lock files, extensions of the name space across file systems, the ability to use arbitrary length file names, and provisions for efficient administrative control of resource usage.

Also described are the changes between the original 512 byte UNIX file system to the file system implemented with the first Berkeley-compatible release of Sun's version of the UNIX system. It presents the motivations for the changes, the methods used to affect these changes, the rationale behind the design decisions, and a description of the new implementation. This discussion is followed by a summary of the results that have been obtained, directions for future work, and the additions and changes that have been made to the user visible facilities. The paper concludes with a history of the software engineering of the project.

The original UNIX system that runs on the PDP-11[1] has simple and elegant file system facilities. File system input/output is buffered by the kernel; there are no alignment constraints on data transfers and all operations are made to appear synchronous. All transfers to the disk are in 512 byte blocks, which can be placed arbitrarily within the data area of the file system. No constraints other than available disk space are placed on file growth [Ritchie74], [Thompson79].

When used together with other UNIX enhancements, the original 512 byte UNIX file system is incapable of providing the data throughput rates that many applications require. For example, applications that need to do a small amount of processing on a large quantities of data such as VLSI design and image processing, need to have a high throughput from the file system. High throughput rates are also needed by programs with large address spaces that are constructed by mapping files from the file system into virtual memory. Paging data in and out of the file system is likely to occur frequently. This requires a file system providing

[1] DEC, PDP, VAX, MASSBUS, and UNIBUS are trademarks of Digital Equipment Corporation.

higher bandwidth than the original 512 byte UNIX one which provides only about two percent of the maximum disk bandwidth or about 20 kilobytes per second per arm [White80], [Smith81b].

Modifications have been made to the UNIX file system to improve its performance. Since the UNIX file system interface is well understood and not inherently slow, this development retained the abstraction and simply changed the underlying implementation to increase its throughput. Consequently users of the system have not been faced with massive software conversion.

Problems with file system performance have been dealt with extensively in the literature; see [Smith81a] for a survey. The UNIX operating system drew many of its ideas from Multics, a large, high performance operating system [Feiertag71]. Other work includes Hydra [Almes78], Spice [Thompson80], and a file system for a lisp environment [Symbolics81a].

A major goal of this project has been to build a file system that is extensible into a networked environment [Holler73]. Other work on network file systems describe centralized file servers [Accetta80], distributed file servers [Dion80], [Luniewski77], [Porcar82], and protocols to reduce the amount of information that must be transferred across a network [Symbolics81b], [Sturgis80].

B.1. Old File System

In the old file system developed at Bell Laboratories each disk drive contains one or more file systems[2]. A file system is described by its super-block, which contains the basic parameters of the file system. These include the number of data blocks in the file system, a count of the maximum number of files, and a pointer to a list of free blocks. All the free blocks in the system are chained together in a linked list. Within the file system are files. Certain files are distinguished as directories and contain pointers to files that may themselves be directories. Every file has a descriptor associated with it called an *inode*. The inode contains information describing ownership of the file, time stamps marking last modification and access times for the file, and an array of indices that point to the data blocks for the file. For the purposes of this section, we assume that the first 8 blocks of the file are directly referenced by values stored in the inode structure itself[3]. The inode structure may also contain references to indirect blocks containing further data block indices. In a file system with a 512 byte block size, a singly indirect block contains 128 further block addresses, a doubly indirect block contains 128 addresses of further single indirect blocks, and a triply indirect block contains 128 addresses of further doubly indirect blocks.

A traditional 150 megabyte UNIX file system consists of 4 megabytes of inodes followed by 146 megabytes of data. This organization segregates the inode information from the data; thus accessing a file normally incurs a long seek from its inode to its data. Files in a single directory are not typically allocated slots in consecutive locations in the 4 megabytes of inodes, causing many non-consecutive blocks to be accessed when executing operations on all the files in a directory.

---

[2] A file system always resides on a single drive.

[3] The actual number may vary from system to system, but is usually in the range 5-13.

The allocation of data blocks to files is also suboptimum. The traditional file system never transfers more than 512 bytes per disk transaction and often finds that the next sequential data block is not on the same cylinder, forcing seeks between 512 byte transfers. The combination of the small block size, limited read-ahead in the system, and many seeks severely limits file system throughput.

The first work at Berkeley on the UNIX file system attempted to improve both reliability and throughput. The reliability was improved by changing the file system so that all modifications of critical information were staged so that they could either be completed or repaired cleanly by a program after a crash [Kowalski78]. The file system performance was improved by a factor of more than two by changing the basic block size from 512 to 1024 bytes. The increase was because of two factors; each disk transfer accessed twice as much data, and most files could be described without need to access through any indirect blocks since the direct blocks contained twice as much data. The file system with these changes will henceforth be referred to as the *old file system.*

This performance improvement gave a strong indication that increasing the block size was a good method for improving throughput. Although the throughput had doubled, the old file system was still using only about four percent of the disk bandwidth. The main problem was that although the free list was initially ordered for optimal access, it quickly became scrambled as files were created and removed. Eventually the free list became entirely random causing files to have their blocks allocated randomly over the disk. This forced the disk to seek before every block access. Although old file systems provided transfer rates of up to 175 kilobytes per second when they were first created, this rate deteriorated to 30 kilobytes per second after a few weeks of moderate use because of randomization of their free block list. There was no way of restoring the performance an old file system except to dump, rebuild, and restore the file system. Another possibility would be to have a process that periodically reorganized the data on the disk to restore locality as suggested by [Maruyama76].

B.2. New file system organization

As in the old file system organization each disk drive contains one or more file systems. A file system is described by its super-block, that is located at the beginning of its disk partition. Because the super-block contains critical data it is replicated to protect against catastrophic loss. This is done at the time that the file system is created; since the super-block data does not change, the copies need not be referenced unless a head crash or other hard disk error causes the default super-block to be unusable.

To ensure that it is possible to create files as large as $2\uparrow 32$ bytes with only two levels of indirection, the minimum size of a file system block is 4096 bytes. The size of file system blocks can be any power of two greater than or equal to 4096. The block size of the file system is maintained in the super-block so it is possible for file systems with different block sizes to be accessible simultaneously on the same system. The block size must be decided at the time that the file system is created; it cannot be subsequently changed without rebuilding the file system.

The new file system organization partitions the disk into one or more areas called *cylinder groups*. A cylinder group is comprised of one or more consecutive cylinders on a disk. Associated with each cylinder group is some bookkeeping

information that includes a redundant copy of the super-block, space for inodes, a bit map describing available blocks in the cylinder group, and summary information describing the usage of data blocks within the cylinder group. For each cylinder group a static number of inodes is allocated at file system creation time. The current policy is to allocate one inode for each 2048 bytes of disk space, expecting this to be far more than will ever be needed.

All the cylinder group bookkeeping information could be placed at the beginning of each cylinder group. However if this approach were used, all the redundant information would be on the top platter. Thus a single hardware failure that destroyed the top platter could cause the loss of all copies of the redundant super-blocks. Thus the cylinder group bookkeeping information begins at a floating offset from the beginning of the cylinder group. The offset for each successive cylinder group is calculated to be about one track further from the beginning of the cylinder group. In this way the redundant information spirals down into the pack so that any single track, cylinder, or platter can be lost without losing all copies of the super-blocks. Except for the first cylinder group, the space between the beginning of the cylinder group and the beginning of the cylinder group information is used for data blocks[4].

Optimizing storage utilization

Data is laid out so that larger blocks can be transferred in a single disk transfer, greatly increasing file system throughput. As an example, consider a file in the new file system composed of 4096 byte data blocks. In the old file system this file would be composed of 1024 byte blocks. By increasing the block size, disk accesses in the new file system may transfer up to four times as much information per disk transaction. In large files, several 4096 byte blocks may be allocated from the same cylinder so that even larger data transfers are possible before initiating a seek.

The main problem with bigger blocks is that most UNIX file systems are composed of many small files. A uniformly large block size wastes space. Table 1 shows the effect of file system block size on the amount of wasted space in the file system. The machine measured to obtain these figures is one of our time sharing systems that has roughly 1.2 Gigabyte of on-line storage. The measurements are based on the active user file systems containing about 920 megabytes of formated space.

[4] While it appears that the first cylinder group could be laid out with its super-block at the "known" location, this would not work for file systems with blocks sizes of 16K or greater, because of the requirement that the cylinder group information must begin at a block boundary.

Table B-1 *Wasted Space as a function of Block Size*

Space used	% waste	Organization
775.2 Mb	0.0	Data only, no separation between files
807.8 Mb	4.2	Data only, each file starts on 512 byte boundary
828.7 Mb	6.9	512 byte block UNIX file system
866.5 Mb	11.8	1024 byte block UNIX file system
948.5 Mb	22.4	2048 byte block UNIX file system
1128.3 Mb	45.6	4096 byte block UNIX file system

The space wasted is measured as the percentage of space on the disk not containing user data. As the block size on the disk increases, the waste rises quickly, to an intolerable 45.6% waste with 4096 byte file system blocks.

To be able to use large blocks without undue waste, small files must be stored in a more efficient way. The new file system accomplishes this goal by allowing the division of a single file system block into one or more *fragments*. The file system fragment size is specified at the time that the file system is created; each file system block can be optionally broken into 2, 4, or 8 fragments, each of which is addressable. The lower bound on the size of these fragments is constrained by the disk sector size, typically 512 bytes. The block map associated with each cylinder group records the space availability at the fragment level; to determine block availability, aligned fragments are examined. Figure 1 shows a piece of a map from a 4096/1024 file system.

Bits in map	XXXX	XXOO	OOXX	OOOO
Fragment numbers	0-3	4-7	8-11	12-15
Block numbers	0	1	2	3

Figure B-1 *Example layout of blocks and fragments in a 4096/1024 file system*

Each bit in the map records the status of a fragment; an "X" shows that the fragment is in use, while a "O" shows that the fragment is available for allocation. In this example, fragments 0–5, 10, and 11 are in use, while fragments 6–9, and 12–15 are free. Fragments of adjoining blocks cannot be used as a block, even if they are large enough. In this example, fragments 6–9 cannot be coalesced into a block; only fragments 12–15 are available for allocation as a block.

On a file system with a block size of 4096 bytes and a fragment size of 1024 bytes, a file is represented by zero or more 4096 byte blocks of data, and possibly a single fragmented block. If a file system block must be fragmented to obtain space for a small amount of data, the remainder of the block is made available for allocation to other files. As an example consider an 11000 byte file stored on a 4096/1024 byte file system. This file would uses two full size blocks and a 3072 byte fragment. If no 3072 byte fragments are available at the time the file is

created, a full size block is split yielding the necessary 3072 byte fragment and an unused 1024 byte fragment. This remaining fragment can be allocated to another file as needed.

The granularity of allocation is the *write* system call. Each time data is written to a file, the system checks to see if the size of the file has increased[5]. If the file needs to hold the new data, one of three conditions exists:

1) There is enough space left in an already allocated block to hold the new data. The new data is written into the available space in the block.

2) Nothing has been allocated. If the new data contains more than 4096 bytes, a 4096 byte block is allocated and the first 4096 bytes of new data is written there. This process is repeated until less than 4096 bytes of new data remain. If the remaining new data to be written will fit in three or fewer 1024 byte pieces, an unallocated fragment is located, otherwise a 4096 byte block is located. The new data is written into the located piece.

3) A fragment has been allocated. If the number of bytes in the new data plus the number of bytes already in the fragment exceeds 4096 bytes, a 4096 byte block is allocated. The contents of the fragment is copied to the beginning of the block and the remainder of the block is filled with the new data. The process then continues as in (2) above. If the number of bytes in the new data plus the number of bytes already in the fragment will fit in three or fewer 1024 byte pieces, an unallocated fragment is located, otherwise a 4096 byte block is located. The contents of the previous fragment appended with the new data is written into the allocated piece.

The problem with allowing only a single fragment on a 4096/1024 byte file system is that data may be potentially copied up to three times as its requirements grow from a 1024 byte fragment to a 2048 byte fragment, then a 3072 byte fragment, and finally a 4096 byte block. The fragment reallocation can be avoided if the user program writes a full block at a time, except for a partial block at the end of the file. Because file systems with different block sizes may coexist on the same system, the file system interface been extended to provide the ability to determine the optimal size for a read or write. For files the optimal size is the block size of the file system on which the file is being accessed. For other objects, such as pipes and sockets, the optimal size is the underlying buffer size. This feature is used by the Standard Input/Output Library, a package used by most user programs. This feature is also used by certain system utilities such as archivers and loaders that do their own input and output management and need the highest possible file system bandwidth.

The space overhead in the 4096/1024 byte new file system organization is empirically observed to be about the same as in the 1024 byte old file system organization. A file system with 4096 byte blocks and 512 byte fragments has about the same amount of space overhead as the 512 byte block UNIX file system. The new file system is more space efficient than the 512 byte or 1024 byte file

[5] A program may be overwriting data in the middle of an existing file in which case space will already be allocated.

systems in that it uses the same amount of space for small files while requiring less indexing information for large files. This savings is offset by the need to use more space for keeping track of available free blocks. The net result is about the same disk utilization when the new file systems fragment size equals the old file systems block size.

In order for the layout policies to be effective, the disk cannot be kept completely full. Each file system maintains a parameter that gives the minimum acceptable percentage of file system blocks that can be free. If the the number of free blocks drops below this level only the system administrator can continue to allocate blocks. The value of this parameter can be changed at any time, even when the file system is mounted and active. The transfer rates to be given in section 4 were measured on file systems kept less than 90% full. If the reserve of free blocks is set to zero, the file system throughput rate tends to be cut in half, because of the inability of the file system to localize the blocks in a file. If the performance is impaired because of overfilling, it may be restored by removing enough files to obtain 10% free space. Access speed for files created during periods of little free space can be restored by recreating them once enough space is available. The amount of free space maintained must be added to the percentage of waste when comparing the organizations given in Table 1. Thus, a site running the old 1024 byte UNIX file system wastes 11.8% of the space and one could expect to fit the same amount of data into a 4096/512 byte new file system with 5% free space, since a 512 byte old file system wasted 6.9% of the space.

File system parameterization

Except for the initial creation of the free list, the old file system ignores the parameters of the underlying hardware. It has no information about either the physical characteristics of the mass storage device, or the hardware that interacts with it. A goal of the new file system is to parameterize the processor capabilities and mass storage characteristics so that blocks can be allocated in an optimum configuration dependent way. Parameters used include the speed of the processor, the hardware support for mass storage transfers, and the characteristics of the mass storage devices. Disk technology is constantly improving and a given installation can have several different disk technologies running on a single processor. Each file system is parameterized so that it can adapt to the characteristics of the disk on which it is placed.

For mass storage devices such as disks, the new file system tries to allocate new blocks on the same cylinder as the previous block in the same file. Optimally, these new blocks will also be well positioned rotationally. The distance between "rotationally optimal" blocks varies greatly; it can be a consecutive block or a rotationally delayed block depending on system characteristics. On a processor with a channel that does not require any processor intervention between mass storage transfer requests, two consecutive disk blocks often can be accessed without suffering lost time because of an intervening disk revolution. For processors without such channels, the main processor must field an interrupt and prepare for a new disk transfer. The expected time to service this interrupt and schedule a new disk transfer depends on the speed of the main processor.

The physical characteristics of each disk include the number of blocks per track and the rate at which the disk spins. The allocation policy routines use this

information to calculate the number of milliseconds required to skip over a block. The characteristics of the processor include the expected time to schedule an interrupt. Given the previous block allocated to a file, the allocation routines calculate the number of blocks to skip over so that the next block in a file will be coming into position under the disk head in the expected amount of time that it takes to start a new disk transfer operation. For programs that sequentially access large amounts of data, this strategy minimizes the amount of time spent waiting for the disk to position itself.

To ease the calculation of finding rotationally optimal blocks, the cylinder group summary information includes a count of the availability of blocks at different rotational positions. Eight rotational positions are distinguished, so the resolution of the summary information is 2 milliseconds for a typical 3600 revolution per minute drive.

The parameter that defines the minimum number of milliseconds between the completion of a data transfer and the initiation of another data transfer on the same cylinder can be changed at any time, even when the file system is mounted and active. If a file system is parameterized to lay out blocks with rotational separation of 2 milliseconds, and the disk pack is then moved to a system that has a processor requiring 4 milliseconds to schedule a disk operation, the throughput will drop precipitously because of lost disk revolutions on nearly every block. If the eventual target machine is known, the file system can be parameterized for it even though it is initially created on a different processor. Even if the move is not known in advance, the rotational layout delay can be reconfigured after the disk is moved so that all further allocation is done based on the characteristics of the new host.

Layout policies

The file system policies are divided into two distinct parts. At the top level are global policies that use file system wide summary information to make decisions regarding the placement of new inodes and data blocks. These routines are responsible for deciding the placement of new directories and files. They also calculate rotationally optimal block layouts, and decide when to force a long seek to a new cylinder group because there are insufficient blocks left in the current cylinder group to do reasonable layouts. Below the global policy routines are the local allocation routines that use a locally optimal scheme to lay out data blocks.

Two methods for improving file system performance are to increase the locality of reference to minimize seek latency as described by [Trivedi80], and to improve the layout of data to make larger transfers possible as described by [Nevalainen77]. The global layout policies try to improve performance by clustering related information. They cannot attempt to localize all data references, but must also try to spread unrelated data among different cylinder groups. If too much localization is attempted, the local cylinder group may run out of space forcing the data to be scattered to non-local cylinder groups. Taken to an extreme, total localization can result in a single huge cluster of data resembling the old file system. The global policies try to balance the two conflicting goals of localizing data that is concurrently accessed while spreading out unrelated data.

One allocatable resource is inodes. Inodes are used to describe both files and directories. Files in a directory are frequently accessed together. For example

the "list directory" command often accesses the inode for each file in a directory. The layout policy tries to place all the files in a directory in the same cylinder group. To ensure that files are allocated throughout the disk, a different policy is used for directory allocation. A new directory is placed in the cylinder group that has a greater than average number of free inodes, and the fewest number of directories in it already. The intent of this policy is to allow the file clustering policy to succeed most of the time. The allocation of inodes within a cylinder group is done using a next free strategy. Although this allocates the inodes randomly within a cylinder group, all the inodes for each cylinder group can be read with 4 to 8 disk transfers. This puts a small and constant upper bound on the number of disk transfers required to access all the inodes for all the files in a directory as compared to the old file system where typically, one disk transfer is needed to get the inode for each file in a directory.

The other major resource is the data blocks. Since data blocks for a file are typically accessed together, the policy routines try to place all the data blocks for a file in the same cylinder group, preferably rotationally optimally on the same cylinder. The problem with allocating all the data blocks in the same cylinder group is that large files will quickly use up available space in the cylinder group, forcing a spill over to other areas. Using up all the space in a cylinder group has the added drawback that future allocations for any file in the cylinder group will also spill to other areas. Ideally none of the cylinder groups should ever become completely full. The solution devised is to redirect block allocation to a newly chosen cylinder group when a file exceeds 32 kilobytes, and at every megabyte thereafter. The newly chosen cylinder group is selected from those cylinder groups that have a greater than average number of free blocks left. Although big files tend to be spread out over the disk, a megabyte of data is typically accessible before a long seek must be performed, and the cost of one long seek per megabyte is small.

The global policy routines call local allocation routines with requests for specific blocks. The local allocation routines will always allocate the requested block if it is free. If the requested block is not available, the allocator allocates a free block of the requested size that is rotationally closest to the requested block. If the global layout policies had complete information, they could always request unused blocks and the allocation routines would be reduced to simple bookkeeping. However, maintaining complete information is costly; thus the implementation of the global layout policy uses heuristic guesses based on partial information.

If a requested block is not available the local allocator uses a four level allocation strategy:

1) Use the available block rotationally closest to the requested block on the same cylinder.

2) If there are no blocks available on the same cylinder, use a block within the same cylinder group.

3) If the cylinder group is entirely full, quadratically rehash among the cylinder groups looking for a free block.

4) Finally if the rehash fails, apply an exhaustive search.

The use of quadratic rehash is prompted by studies of symbol table strategies used in programming languages. File systems that are parameterized to maintain at least 10% free space almost never use this strategy; file systems that are run without maintaining any free space typically have so few free blocks that almost any allocation is random. Consequently the most important characteristic of the strategy used when the file system is low on space is that it be fast.

B.3. Performance

Ultimately, the proof of the effectiveness of the algorithms described in the previous section is the long term performance of the new file system.

Our empiric studies have shown that the inode layout policy has been effective. When running the "list directory" command on a large directory that itself contains many directories, the number of disk accesses for inodes is cut by a factor of two. The improvements are even more dramatic for large directories containing only files, disk accesses for inodes being cut by a factor of eight. This is most encouraging for programs such as spooling daemons that access many small files, since these programs tend to flood the disk request queue on the old file system.

Table 2 summarizes the measured throughput of the new file system. Several comments need to be made about the conditions under which these tests were run. The test programs measure the rate that user programs can transfer data to or from a file without performing any processing on it. These programs must write enough data to ensure that buffering in the operating system does not affect the results. They should also be run at least three times in succession; the first to get the system into a known state and the second two to ensure that the experiment has stabilized and is repeatable. The methodology and test results are discussed in detail in [Kridle83][6]. The systems were running multi-user but were otherwise quiescent. There was no contention for either the cpu or the disk arm. The only difference between the UNIBUS and MASSBUS tests was the controller. All tests used an Ampex Capricorn 330 Megabyte Winchester disk. As Table 2 shows, all file system test runs were on a VAX 11/750. All file systems had been in production use for at least a month before being measured.

[6] A UNIX command that is similar to the reading test that we used is, "cp file /dev/null", where "file" is eight Megabytes long.

Table B-2 *Reading Rates of the Old and New UNIX File Systems*

Type of File System	Processor and Bus Measured	Speed	Read Bandwidth	% CPU
old 1024	750/UNIBUS	29 Kbytes/sec	29/1100 3%	11%
new 4096/1024	750/UNIBUS	221 Kbytes/sec	221/1100 20%	43%
new 8192/1024	750/UNIBUS	233 Kbytes/sec	233/1100 21%	29%
new 4096/1024	750/MASSBUS	466 Kbytes/sec	466/1200 39%	73%
new 8192/1024	750/MASSBUS	466 Kbytes/sec	466/1200 39%	54%

Table B-3 *Writing rates of the old and new UNIX file systems*

Type of File System	Processor and Bus Measured	Speed	Write Bandwidth	% CPU
old 1024	750/UNIBUS	48 Kbytes/sec	48/1100 4%	29%
new 4096/1024	750/UNIBUS	142 Kbytes/sec	142/1100 13%	43%
new 8192/1024	750/UNIBUS	215 Kbytes/sec	215/1100 19%	46%
new 4096/1024	750/MASSBUS	323 Kbytes/sec	323/1200 27%	94%
new 8192/1024	750/MASSBUS	466 Kbytes/sec	466/1200 39%	95%

Unlike the old file system, the transfer rates for the new file system do not appear to change over time. The throughput rate is tied much more strongly to the amount of free space that is maintained. The measurements in Table 2 were based on a file system run with 10% free space. Synthetic work loads suggest the performance deteriorates to about half the throughput rates given in Table 2 when no free space is maintained.

The percentage of bandwidth given in Table 2 is a measure of the effective utilization of the disk by the file system. An upper bound on the transfer rate from the disk is measured by doing 65536[7] byte reads from contiguous tracks on the disk. The bandwidth is calculated by comparing the data rates the file system is able to achieve as a percentage of this rate. Using this metric, the old file system is only able to use about 3-4% of the disk bandwidth, while the new file system uses up to 39% of the bandwidth.

In the new file system, the reading rate is always at least as fast as the writing rate. This is to be expected since the kernel must do more work when allocating blocks than when simply reading them. Note that the write rates are about the same as the read rates in the 8192 byte block file system; the write rates are slower than the read rates in the 4096 byte block file system. The slower write

[7] This number, 65536, is the maximal I/O size supported by the VAX hardware; it is a remnant of the system's PDP-11 ancestry.

rates occur because the kernel has to do twice as many disk allocations per second, and the processor is unable to keep up with the disk transfer rate.

In contrast the old file system is about 50% faster at writing files than reading them. This is because the *write* system call is asynchronous and the kernel can generate disk transfer requests much faster than they can be serviced, hence disk transfers build up in the disk buffer cache. Because the disk buffer cache is sorted by minimum seek order, the average seek between the scheduled disk writes is much less than they would be if the data blocks are written out in the order in which they are generated. However when the file is read, the *read* system call is processed synchronously so the disk blocks must be retrieved from the disk in the order in which they are allocated. This forces the disk scheduler to do long seeks resulting in a lower throughput rate.

The performance of the new file system is currently limited by a memory to memory copy operation because it transfers data from the disk into buffers in the kernel address space and then spends 40% of the processor cycles copying these buffers to user address space. If the buffers in both address spaces are properly aligned, this transfer can be affected without copying by using the VAX virtual memory management hardware. This is especially desirable when large amounts of data are to be transferred. We did not implement this because it would change the semantics of the file system in two major ways; user programs would be required to allocate buffers on page boundaries, and data would disappear from buffers after being written.

Greater disk throughput could be achieved by rewriting the disk drivers to chain together kernel buffers. This would allow files to be allocated to contiguous disk blocks that could be read in a single disk transaction. Most disks contain either 32 or 48 512 byte sectors per track. The inability to use contiguous disk blocks effectively limits the performance on these disks to less than fifty percent of the available bandwidth. Since each track has a multiple of sixteen sectors it holds exactly two or three 8192 byte file system blocks, or four or six 4096 byte file system blocks. If the the next block for a file cannot be laid out contiguously, then the minimum spacing to the next allocatable block on any platter is between a sixth and a half a revolution. The implication of this is that the best possible layout without contiguous blocks uses only half of the bandwidth of any given track. If each track contains an odd number of sectors, then it is possible to resolve the rotational delay to any number of sectors by finding a block that begins at the desired rotational position on another track. The reason that block chaining has not been implemented is because it would require rewriting all the disk drivers in the system, and the current throughput rates are already limited by the speed of the available processors.

Currently only one block is allocated to a file at a time. A technique used by the DEMOS file system when it finds that a file is growing rapidly, is to preallocate several blocks at once, releasing them when the file is closed if they remain unused. By batching up the allocation the system can reduce the overhead of allocating at each write, and it can cut down on the number of disk writes needed to keep the block pointers on the disk synchronized with the block allocation [Powell79].

B.4. File system functional enhancements

The speed enhancements to the UNIX file system did not require any changes to the semantics or data structures viewed by the users. However several changes have been generally desired for some time but have not been introduced because they would require users to dump and restore all their file systems. Since the new file system already requires that all existing file systems be dumped and restored, these functional enhancements have been introduced at this time.

Long file names

File names can now be of nearly arbitrary length. The only user programs affected by this change are those that access directories. To maintain portability among UNIX systems that are not running the new file system, a set of directory access routines have been introduced that provide a uniform interface to directories on both old and new systems.

Directories are allocated in units of 512 bytes. This size is chosen so that each allocation can be transferred to disk in a single atomic operation. Each allocation unit contains variable-length directory entries. Each entry is wholly contained in a single allocation unit. The first three fields of a directory entry are fixed and contain an inode number, the length of the entry, and the length of the name contained in the entry. Following this fixed size information is the null terminated name, padded to a 4 byte boundary. The maximum length of a name in a directory is currently 255 characters.

Free space in a directory is held by entries that have a record length that exceeds the space required by the directory entry itself. All the bytes in a directory unit are claimed by the directory entries. This normally results in the last entry in a directory being large. When entries are deleted from a directory, the space is returned to the previous entry in the same directory unit by increasing its length. If the first entry of a directory unit is free, then its inode number is set to zero to show that it is unallocated.

File locking

The old file system had no provision for locking files. Processes that needed to synchronize the updates of a file had to create a separate "lock" file to synchronize their updates. A process would try to create a "lock" file. If the creation succeeded, then it could proceed with its update; if the creation failed, then it would wait, and try again. This mechanism had three drawbacks. Processes consumed CPU time, by looping over attempts to create locks. Locks were left lying around following system crashes and had to be cleaned up by hand. Finally, processes running as system administrator are always permitted to create files, so they had to use a different mechanism. While it is possible to get around all these problems, the solutions are not straight-forward, so a mechanism for locking files has been added.

The most general schemes allow processes to concurrently update a file. Several of these techniques are discussed in [Peterson83]. A simpler technique is to simply serialize access with locks. To attain reasonable efficiency, certain applications require the ability to lock pieces of a file. Locking down to the byte level has been implemented in the Onyx file system by [Bass81]. However, for the applications that currently run on the system, a mechanism that locks at the granularity of a file is sufficient.

Locking schemes fall into two classes, those using hard locks and those using advisory locks. The primary difference between advisory locks and hard locks is the decision of when to override them. A hard lock is always enforced whenever a program tries to access a file; an advisory lock is only applied when it is requested by a program. Thus advisory locks are only effective when all programs accessing a file use the locking scheme. With hard locks there must be some override policy implemented in the kernel, with advisory locks the policy is implemented by the user programs. In the UNIX system, programs with system administrator privilege can override any protection scheme. Because many of the programs that need to use locks run as system administrators, we chose to implement advisory locks rather than create a protection scheme that was contrary to the UNIX philosophy or could not be used by system administration programs.

The file locking facilities allow cooperating programs to apply advisory *shared* or *exclusive* locks on files. Only one process has an exclusive lock on a file while multiple shared locks may be present. Both shared and exclusive locks cannot be present on a file at the same time. If any lock is requested when another process holds an exclusive lock, or an exclusive lock is requested when another process holds any lock, the open will block until the lock can be gained. Because shared and exclusive locks are advisory only, even if a process has obtained a lock on a file, another process can override the lock by opening the same file without a lock.

Locks can be applied or removed on open files, so that locks can be manipulated without needing to close and reopen the file. This is useful, for example, when a process wishes to open a file with a shared lock to read some information, to determine whether an update is required. It can then get an exclusive lock so that it can do a read, modify, and write to update the file in a consistent manner.

A request for a lock will cause the process to block if the lock can not be immediately obtained. In certain instances this is unsatisfactory. For example, a process that wants only to check if a lock is present would require a separate mechanism to find out this information. Consequently, a process may specify that its locking request should return with an error if a lock can not be immediately obtained. Being able to poll for a lock is useful to ''daemon'' processes that wish to service a spooling area. If the first instance of the daemon locks the directory where spooling takes place, later daemon processes can easily check to see if an active daemon exists. Since the lock is removed when the process exits or the system crashes, there is no problem with unintentional locks files that must be cleared by hand.

Almost no deadlock detection is attempted. The only deadlock detection made by the system is that the file descriptor to which a lock is applied does not currently have a lock of the same type (i.e. the second of two successive calls to apply a lock of the same type will fail). Thus a process can deadlock itself by requesting locks on two separate file descriptors for the same object.

Symbolic links

The 512 byte UNIX file system allows multiple directory entries in the same file system to reference a single file. The link concept is fundamental; files do not live in directories, but exist separately and are referenced by links. When all the links are removed, the file is deallocated. This style of links does not allow references across physical file systems, nor does it support inter-machine linkage. To avoid these limitations *symbolic links* have been added similar to the scheme used by Multics [Feiertag71].

A symbolic link is implemented as a file that contains a pathname. When the system encounters a symbolic link while interpreting a component of a pathname, the contents of the symbolic link is prepended to the rest of the pathname, and this name is interpreted to yield the resulting pathname. If the symbolic link contains an absolute pathname, the absolute pathname is used, otherwise the contents of the symbolic link is evaluated relative to the location of the link in the file hierarchy.

Normally programs do not want to be aware that there is a symbolic link in a pathname that they are using. However certain system utilities must be able to detect and manipulate symbolic links. Three new system calls provide the ability to detect, read, and write symbolic links, and seven system utilities were modified to use these calls.

In future Berkeley software distributions it will be possible to mount file systems from other machines within a local file system. When this occurs, it will be possible to create symbolic links that span machines.

Rename

Programs that create new versions of data files typically create the new version as a temporary file and then rename the temporary file with the original name of the data file. In the old UNIX file systems the renaming required three calls to the system. If the program were interrupted or the system crashed between these calls, the data file could be left with only its temporary name. To eliminate this possibility a single system call has been added that performs the rename in an atomic fashion to guarantee the existence of the original name.

In addition, the rename facility allows directories to be moved around in the directory tree hierarchy. The rename system call performs special validation checks to ensure that the directory tree structure is not corrupted by the creation of loops or inaccessible directories. Such corruption would occur if a parent directory were moved into one of its descendants. The validation check requires tracing the ancestry of the target directory to ensure that it does not include the directory being moved.

Quotas

The UNIX system has traditionally attempted to share all available resources to the greatest extent possible. Thus any single user can allocate all the available space in the file system. In certain environments this is unacceptable. Consequently, a quota mechanism has been added for restricting the amount of file system resources that a user can obtain. The quota mechanism sets limits on both the number of files and the number of disk blocks that a user may allocate. A separate quota can be set for each user on each file system. Each resource is given both a hard and a soft limit. When a program exceeds a soft limit, a warning is printed on the users terminal; the offending program is not terminated

unless it exceeds its hard limit. The idea is that users should stay below their soft limit between login sessions, but they may use more space while they are actively working. To encourage this behavior, users are warned when logging in if they are over any of their soft limits. If they fail to correct the problem for too many login sessions, they are eventually reprimanded by having their soft limit enforced as their hard limit.

B.5. Software engineering

The preliminary design was done by Bill Joy in late 1980; he presented the design at The USENIX Conference held in San Francisco in January 1981. The implementation of his design was done by Kirk McKusick in the summer of 1981. Most of the new system calls were implemented by Sam Leffler. The code for enforcing quotas was implemented by Robert Elz at the University of Melbourne.

To understand how the project was done it is necessary to understand the interfaces that the UNIX system provides to the hardware mass storage systems. At the lowest level is a *raw disk*. This interface provides access to the disk as a linear array of sectors. Normally this interface is only used by programs that need to do disk to disk copies or that wish to dump file systems. However, user programs with proper access rights can also access this interface. A disk is usually formated with a file system that is interpreted by the UNIX system to provide a directory hierarchy and files. The UNIX system interprets and multiplexes requests from user programs to create, read, write, and delete files by allocating and freeing inodes and data blocks. The interpretation of the data on the disk could be done by the user programs themselves. The reason that it is done by the UNIX system is to synchronize the user requests, so that two processes do not attempt to allocate or modify the same resource simultaneously. It also allows access to be restricted at the file level rather than at the disk level and allows the common file system routines to be shared between processes.

The implementation of the new file system amounted to using a different scheme for formating and interpreting the disk. Since the synchronization and disk access routines themselves were not being changed, the changes to the file system could be developed by moving the file system interpretation routines out of the kernel and into a user program. Thus, the first step was to extract the file system code for the old file system from the UNIX kernel and change its requests to the disk driver to accesses to a raw disk. This produced a library of routines that mapped what would normally be system calls into read or write operations on the raw disk. This library was then debugged by linking it into the system utilities that copy, remove, archive, and restore files.

A new cross file system utility was written that copied files from the simulated file system to the one implemented by the kernel. This was accomplished by calling the simulation library to do a read, and then writing the resultant data by using the conventional write system call. A similar utility copied data from the kernel to the simulated file system by doing a conventional read system call and then writing the resultant data using the simulated file system library.

The second step was to rewrite the file system simulation library to interpret the new file system. By linking the new simulation library into the cross file system copying utility, it was possible to easily copy files from the old file system into

the new one and from the new one to the old one. Having the file system interpretation implemented in user code had several major benefits. These included being able to use the standard system tools such as the debuggers to set breakpoints and single step through the code. When bugs were discovered, the offending problem could be fixed and tested without the need to reboot the machine. There was never a period where it was necessary to maintain two concurrent file systems in the kernel. Finally it was not necessary to dedicate a machine entirely to file system development, except for a brief period while the new file system was boot strapped.

The final step was to merge the new file system back into the UNIX kernel. This was done in less than two weeks, since the only bugs remaining were those that involved interfacing to the synchronization routines that could not be tested in the simulated system. Again the simulation system proved useful since it enabled files to be easily copied between old and new file systems regardless of which file system was running in the kernel. This greatly reduced the number of times that the system had to be rebooted.

The total design and debug time took about one man year. Most of the work was done on the file system utilities, and changing all the user programs to use the new facilities. The code changes in the kernel were minor, involving the addition of only about 800 lines of code.

B.6. Acknowledgements

We thank Robert Elz for his ongoing interest in the new file system, and for adding disk quotas in a rational and efficient manner. We also acknowledge Dennis Ritchie for his suggestions on the appropriate modifications to the user interface. We appreciate Michael Powell's explanations on how the DEMOS file system worked; many of his ideas were used in this implementation. Special commendation goes to Peter Kessler and Robert Henry for acting like real users during the early debugging stage when files were less stable than they should have been. Finally we thank our sponsors, the National Science Foundation under grant MCS80-05144, and the Defense Advance Research Projects Agency (DoD) under Arpa Order No. 4031 monitored by Naval Electronic System Command under Contract No. N00039-82-C-0235.

B.7. References

[Accetta80] Accetta, M., Robertson, G., Satyanarayanan, M., and Thompson, M. "The Design of a Network-Based Central File System", Carnegie-Mellon University, Dept of Computer Science Tech Report, #CMU-CS-80-134

[Almes78] Almes, G., and Robertson, G. "An Extensible File System for Hydra" Proceedings of the Third International Conference on Software Engineering, IEEE, May 1978.

[Bass81] Bass, J. "Implementation Description for File Locking", Onyx Systems Inc, 73 E. Trimble Rd, San Jose, CA 95131 Jan 1981.

[Dion80] Dion, J. "The Cambridge File Server", Operating Systems Review, 14, 4. Oct 1980. pp 26-35

[Eswaran74] Eswaran, K. "Placement of records in a file and file allocation in a computer network", Proceedings IFIPS, 1974. pp 304-307

[Holler73] Holler, J. "Files in Computer Networks", First European Workshop on Computer Networks, April 1973. pp 381-396

[Feiertag71] Feiertag, R. J. and Organick, E. I., "The Multics Input-Output System", Proceedings of the Third Symposium on Operating Systems Principles, ACM, Oct 1971. pp 35-41

[Kridle83] Kridle, R., and McKusick, M., "Performance Effects of Disk Subsystem Choices for VAX Systems Running 4.2BSD UNIX", Computer Systems Research Group, Dept of EECS, Berkeley, CA 94720, Technical Report #8.

[Kowalski78] Kowalski, T. "FSCK - The UNIX System Check Program", Bell Laboratory, Murray Hill, NJ 07974. March 1978

[Luniewski77] Luniewski, A. "File Allocation in a Distributed System", MIT Laboratory for Computer Science, Dec 1977.

[Maruyama76] Maruyama, K., and Smith, S. "Optimal reorganization of Distributed Space Disk Files", Communications of the ACM, 19, 11. Nov 1976. pp 634-642

[Nevalainen77] Nevalainen, O., Vesterinen, M. "Determining Blocking Factors for Sequential Files by Heuristic Methods", The Computer Journal, 20, 3. Aug 1977. pp 245-247

[Peterson83] Peterson, G. "Concurrent Reading While Writing", ACM Transactions on Programming Languages and Systems, ACM, 5, 1. Jan 1983. pp 46-55

[Powell79] Powell, M. "The DEMOS File System", Proceedings of the Sixth Symposium on Operating Systems Principles, ACM, Nov 1977. pp 33-42

[Porcar82] Porcar, J. "File Migration in Distributed Computer Systems", Ph.D. Thesis, Lawrence Berkeley Laboratory Tech Report #LBL-14763.

[Ritchie74] Ritchie, D. M. and Thompson, K., "The UNIX Time-Sharing System", CACM 17, 7. July 1974. pp 365-375

[Smith81a] Smith, A. "Input/Output Optimization and Disk Architectures: A Survey", Performance and Evaluation 1. Jan 1981. pp 104-117

[Smith81b] Smith, A. "Bibliography on File and I/O System Optimization and Related Topics", Operating Systems

Review, 15, 4. Oct 1981. pp 39-54

[Sturgis80] Sturgis, H., Mitchell, J., and Israel, J. "Issues in the Design and Use of a Distributed File System", Operating Systems Review, 14, 3. pp 55-79

[Symbolics81a] "Symbolics File System", Symbolics Inc, 9600 DeSoto Ave, Chatsworth, CA 91311 Aug 1981.

[Symbolics81b] "Chaosnet FILE Protocol". Symbolics Inc, 9600 DeSoto Ave, Chatsworth, CA 91311 Sept 1981.

[Thompson79] Thompson, K. "UNIX Implementation", Section 31, Volume 2B, UNIX Programmers Manual, Bell Laboratory, Murray Hill, NJ 07974. Jan 1979

[Thompson80] Thompson, M. "Spice File System", Carnegie-Mellon University, Dept of Computer Science Tech Report, #CMU-CS-80-???

[Trivedi80] Trivedi, K. "Optimal Selection of CPU Speed, Device Capabilities, and File Assignments", Journal of the ACM, 27, 3. July 1980. pp 457-473

[White80] White, R. M. "Disk Storage Technology", Scientific American, 243(2), August 1980.

UUCP Implementation Description

C

UUCP Implementation Description

UUCP Implementation Description **305**

C.1. UUCP — UNIX to UNIX File Copy 305
Type 1 — Local Copy 307
Type 2 — Receive Files 307
Type 3 — Send Files 307
Type 4 and Type 5 — Remote UUCP Required 308
Type 6 — Remote Execution 308
C.2. UUX — UNIX to UNIX Execution 308
User Line 309
Required File Line 309
Standard Input Line 310
Standard Output Line 310
Command Line 310
C.3. UUXQT — UUCP Command Execution 310
C.4. UUCICO — Copy In, Copy Out 310
Scan For Work 311
Call Remote System 312
Line Protocol Selection 312
Work Processing 313
Conversation Termination 314
C.5. UULOG — UUCP Log Inquiry 314
C.6. UUCLEAN — UUCP Spool Directory Cleanup 314
C.7. Security 315

C.8. UUCP Installation 315
uucp.h Modification 316
Makefile Modification 316
Compiling the System 317
Files Required for Execution 317
L-devices — Call Unit Information File 317
L-dialcodes — Dial Code Abbreviations File 318
USERFILE 318
L.sys 320
L.cmds 322
Device Types 322
C.9. Administration 323
SQFILE — Sequence Check File 323
TM — Temporary Data Files 323
LOG — Log Entry Files 323
STST — System Status Files 324
LCK — Lock Files 324
Shell Files 324
Login Entry 325
File Modes 325

C

UUCP Implementation Description

Uucp is a series of programs designed such that UNIX systems can communicate with each other using either dial-up or hardwired communication lines. *Uucp* transfers files between UNIX systems, and can also run commands on remote machines. This document gives a detailed description of the current implementation of *uucp*. It is designed for use by an administrator or installer of the system. It is not meant as a user's guide.

Uucp is a batch operation. Files are created in a spool directory for processing by the *uucp* daemons. There are three types of files used for the execution of work: *Data files* contain data for transfer to remote systems; *work files* contain directions for file transfers between systems; *execute files* are scripts for UNIX commands that involve the resources of one or more systems.

There are four primary programs involved in *uucp*'s operation:

uucp	builds *work files* and gathers *data files* in the spool directory for data transmission.
uux	creates *work files* and *execute files*, and gathers *data files* for the remote execution of UNIX commands.
uucico	executes the work files for data transmission.
uuxqt	executes the scripts for UNIX command execution.

There are two administrative programs:

uulog	gathers temporary log files that may occur due to lockout of the *uucp* log file and reports some information such as copy requests and completion status.
uuclean	removes old files from the spool directory.

The remaining sections of this manual describe the operation of each program, security, installation details, files required for execution, and administration of the *uucp* system.

C.1. UUCP — UNIX to UNIX File Copy

The *uucp* command was is designed to look like *cp* (1) to the user. The syntax is:

uucp [option] ... source ... destination

where the source and destination may contain the prefix *system-name!*, which indicates the system where the file or files reside or where they will be copied.

Uucp has several options:

–d	Make directories when necessary for copying the file.
–c	Don't copy source files to the spool directory, but use the specified source when the actual transfer takes place. Note that the files, and all directories leading to them, must be accessible by everybody.
–e*sys*	Send this job to system *sys* to execute. Note that this only works when the system *sys* allows *uuxqt* to execute a *uucp* command. See the sections entitled *Uuxqt* and *Security*.
–C	Force the source files to be copied to the spool directory.
–f	Do not make directories.
–g*letter*	Put *letter* in as the grade in the name of the work file. This can be used to change the order of work for a particular machine.
–m	Send mail to the requester on completion of the work.
–n*user*	Notify *user* on the remote machine that a file has been sent.

Then there are options available for debugging:

–s*dir*	Use *dir* as the spool directory.
–r	Queue the job but do not start *uucico* program.
–x*num*	*Num* is a level number between 1 and 9; higher numbers give more debugging output.

The destination may be a directory name, in which case the file name is taken from the last part of the source's name. If the directory exists, it must be writable by everybody. Note that if the destination is a directory name and the **–d** option is specified to create the directory, the directory name must be followed by "/". The source name can contain special shell characters such as **?**, *****, and [and]. These are expanded on the appropriate system.

The command

```
uucp   *.c   usg!/usr/dan
```

sets up the transfer of all files whose names end with *.c* to the */usr/dan* directory on the *usg* machine.

The source and/or destination names may also contain a *~user* prefix. This translates to the login directory of *user* on the specified system. File names beginning with "~/" translate into the public directory (usually */usr/spool/uucppublic*) on the remote system. For names with partial pathnames, the current directory is prepended to the file name. File names beginning with ../ are not permitted for security reasons.

The command

```
uucp   usg!~dan/*.h   ~dan
```

sets up the transfer of files whose names end with *.h* in dan's login directory on system *usg* to dan's local login directory.

For each source file, *uucp* checks the source and destination file-names, the system-part of each argument, and the options to classify the work into several types:

[1] Copy source to destination on local system.

[2] Receive files from other systems.

[3] Send files to a remote system.

[4] Send files from remote systems to another remote system.

[5] Receive files from remote systems when the source contains special shell characters as mentioned above.

[6] Request that the *uucp* command be executed by a remote system.

After the work has been set up in the spool directory, the *uucico* program is started to try to contact the other machine and execute the work (unless the –r option was specified).

Type 1 — Local Copy

The copy is done locally. The *–m* and *–n* options are not honored in this case.

Type 2 — Receive Files

A *work file* is created or appended with a one line entry for each request. The upper limit to the number of files per *work file* is set in *uucp.h* The default setting is 20. After the limit has been reached, a new *work file* is created.

All *work files* and *execute files* use a blank as the field separator. The fields for these entries are given below.

[1] R

[2] The full path-name of the source or a ~something/path-name. The *~something* part is expanded on the remote system.

[3] The full path-name of the destination file. If the *~something* notation is used, it is immediately expanded.

[4] The user's login name.

[5] A "–" followed by an option list. The options –m and –d may appear.

Type 3 — Send Files

Each source file is copied into a *data file* in the spool directory. (A –c option on the *uucp* command prevents the *data file* from being made. In this case, the file is transmitted from the indicated source.) The fields for these entries are given below.

[1] S

[2] The full-path name of the source file.

[3] The full-path name of the destination or ~something/file-name.

[4] The user's login name.

[5] A "–" followed by an option list. The options –d, –m, and –n may appear.

[6] The name of the *data file* in the spool directory. A dummy name, "D.0" is used when the –c option is specified.

[7] The file mode bits of the source file in octal print format (666 for instance).

[8] The user on the remote system who should be notified upon completion of the file copy when the –n option is specified.

Type 4 and Type 5 — Remote UUCP Required

Uucp generates a *uucp* command and sends it to the remote machine; the remote *uucico* executes the *uucp* command.

Type 6 — Remote Execution

Remote execution occurs when the **–e** option is used. In this case, the *uux* facility is used to create and send the request. This requires that the remote *uuxqt* program allows the *uucp* command to be executed.

C.2. UUX — UNIX to UNIX Execution

The *uux* command sets up the execution of a UNIX command where the execution machine and/or some of the files are remote. The syntax of the *uux* command is

uux [–] [option] ... command-string

where the command-string is made up of one or more arguments. All special shell characters such as <, >, |, and ^, must be quoted either by quoting the entire command-string or quoting the special character as a separate argument.

Within the command-string, the command and file names may contain a *system-name!* prefix. Arguments which do not contain a **!** character are assumed to be part of the command string and are not treated as files (that is, *uux* does not copy them to the execution machine).

An argument containing a **!** but should not be treated as a file at the present time, can be escaped by surrounding the argument in parentheses (and). Note that the (and) characters themselves must usually be escaped with a \ character.

The "–" indicates that the standard input for *command-string* should be inherited from the standard input of the *uux* command.

The following options are available for *uux*:

–	Read from the standard input.
–x	Read from the standard input.
–n	Do not notify the requestor (by mail) of completion status.
–z	Only notify the requestor (by mail) of non-zero completion status.

The following options are available for debugging:

–r	Don't start *uucico* or *uuxqt* after queuing the job.

–x*num* *Num* is a level number between 1 and 9; higher numbers give more debugging output.

The command:

```
pr  abc  |  uux  -  usg!lpr
```

sets up the output of the

```
pr abc
```

command as standard input to an *lpr* command to be executed on system *usg*.

Uux generates an *execute file* containing the names of the files required for execution (including standard input), the user's login name, the destination of the standard output, and the command to be executed. This *execute file* file is either put in the spool directory for local execution or sent to the remote system using a send command (*uucp* type 3 command, described previously).

For required files that are not on the execution machine, *uux* generates receive command files (*uucp* type 2 command, described previously). The *uucico* program puts these command-files on the execution machine for execution.

The *execute file* contains a script that is processed by the *uuxqt* program. It is made up of several lines, each of which contains an identification character and one or more arguments. The lines are described in the subsections below. Here is a summary of the types of lines that appear in the file. They are described in detail in the sections following.

User Line Identifies the requestor's login name and system.

File Line Identifies a filename for transmission.

Standard Input Line
Specifies a standard input file.

Standard Output Line
Specifies a standard output file.

Command Line
Identifies a UNIX system command for *uuxqt* to execute.

User Line

U user system

where the *user* and *system* are the requester's login name and system.

Required File Line

F file-name real-name

where *file-name* is a unique name used for file transmission and *real-name* is the last part of the actual file name (contains no path information).

Zero or more of these lines may be present. The *uuxqt* program checks for the existence of all these files before the command is executed.

Standard Input Line

I file-name

The standard input is either specified by a < in the command-string or inherited from the standard input of the *uux* command if the "–" option is used. If a standard input is not specified, */dev/null* is used. Note that if there is a standard input specified, it will also appear in an "F" line.

Standard Output Line

O file-name system-name

The standard output is specified by a > within the command-string. If a standard output is not specified, */dev/null* is used. Note that the appending to a file by using >> is not implemented.

Command Line

C command [arguments] ...

The arguments are those specified in the command-string. The standard input and standard output will not appear on this line. All *required files* are moved to the execution directory (usually */usr/lib/uucp/.XQTDIR*) and the UNIX command is executed using the shell specified in the *uucp.h* header file. In addition, a shell "PATH" statement is prepended to the command line as specified in the *uuxqt* program.

Note that a check is made to see that the command is allowed as specified in the *uuxqt* program. After execution, the standard output is copied or sent to the proper place.

C.3. UUXQT — UUCP Command Execution

The *uuxqt* program executes scripts generated by *uux*.

The *uuxqt* program may be started by either the *uucico* or *uux* programs or a daemon specified by a *crontab* entry. *Uuxqt* scans the spool directory for *execute files* (prefix "X."). Each *execute file* is checked to see if all the required files are available and if so, the command line is verified and executed.

The *execute file* is described in the section entitled *uux*, above.

The execution is accomplished by executing a

```
sh -c
```

of the command line after appropriate standard input and standard output have been opened. If a standard output is specified, the program creates a send command, or copies the output file as appropriate.

Uuxqt accepts the standard debugging option:

–x*num* — *Num* is a level number between 1 and 9; higher numbers give more debug output.

C.4. UUCICO — Copy In, Copy Out

The *uucico* program performs several major functions:

- Scan the spool directory for work.
- Place a call to a remote system.
- Negotiate a line protocol to be used.

- Execute all requests from both systems.
- Log work requests and work completions.

Uucico may be started in several ways:

a. by a system daemon specified in a crontab entry,

b. by one of the *uucp, uux, uuxqt* or *uucico* programs,

c. directly by the user (this is usually for testing),

d. by a remote system. The uucico program should be specified as the "shell" field in the */etc/passwd* file for the logins used by remote systems to access uucp.

When started by method a, b or c, *uucico* is considered to be in *MASTER* mode. In this mode, a connection is made to a remote system. If started by a remote system (method d), *uucico* is considered to be in *SLAVE* mode.

The *MASTER* mode operates in one of two ways. If no system name is specified (–s option not specified) *uucico* scans the spool directory for systems to call. If a system name is specified, that system is called, and work is only done for that system.

Uucico is generally started by another program. There are several options used for execution:

–r1 Start *uucico* in *MASTER* mode. This is used when *uucico* is started by a program or "cron" shell.

–s*sys* Do work only for system *sys*. If *–s* is specified, a call to the specified system is made even if there is no work for system *sys* in the spool directory. This is useful for polling systems that do not have the hardware to initiate a connection.

The following options are used primarily for debugging:

–d*dir* Use directory *dir* for the spool directory.

–x*num* *Num* is a level number between 1 and 9; higher numbers give more debugging output.

The next part of this section describes the major steps within the *uucico* program.

Scan For Work

The names of the work related files in the spool directory have format

type . system-name grade number

type is an upper case letter (*C* – copy command file, *D* – data file, *X* – execute file),

system-name
is the remote system, *truncated to seven characters*.

grade is a character,

number is a four digit, hexadecimal, zero padded sequence number.

The file

C.res45n0031

would be a *work file* for a file transfer between the local machine and the "res45" machine.

The scan for work is done by looking through the spool directory for *work files* (files with prefix "C."). A list is made of all systems to be called. *Uucico* then calls each system and processes all *work files*.

Call Remote System

The call is made using information from several files that reside in the *uucp* program directory (usually */usr/lib/uucp*). At the start of the call process, a lock is set to prevent multiple conversations between the same two systems.

The *L.sys* file contains information required to make the remote connection:

[1] system name,

[2] times to call the system (days-of-week and times-of-day) and the minimum time delay before retry,

[3] device or device type to be used for call,

[4] line class (this is the line speed on almost all systems),

[5] phone number if field [3] is *ACU* or the device if not *ACU*,

[6] login information (zero or more fields),

The time field is checked against the present time to see if the call should be made. The *phone number* may contain abbreviations (for example, mh, py, boston) that get translated into dial sequences using the *L-dialcodes* file.

The *L-devices* file is scanned using fields [3] and [4] from the *L.sys* file to find an available device for the call. The program tries each devices that satisfy [3] and [4] until a call is made, or no more devices can be tried. If a device is successfully opened, a lock file is created. If the call is completed, the login information (field [6] of *L.sys)* is used to login.

The conversation between the two *uucico* programs begins with a handshake started by the called (*SLAVE*) system. The *SLAVE* sends a message to let the *MASTER* know it is ready to receive the system identification and conversation sequence number. The *SLAVE* verifies the response from the *MASTER* and if acceptable, protocol selection begins. The *SLAVE* can also reply with a "call-back required" message, in which case the current conversation is terminated.

Line Protocol Selection

The remote system sends a message:

P*proto-list*

where *proto-list* is a string of characters, each representing a line protocol.

The calling program checks *proto-list* for a letter corresponding to an available line protocol and returns a *use-protocol* message. The *use-protocol* message is

U*code*

where *code* is either a one character protocol letter or "N", which means there is no common protocol. The only protocol which is currently implemented is "g", which uses the packet driver.

Work Processing

The *MASTER* program does a work search similar to the one used in the *Scan For Work* section described above (the *MASTER* has been specified by the "–r1" uucico option). Each message used during the work processing is specified by the first character of the message:

S send a file,

R receive a file,

C copy complete,

X execute a *uucp* command,

H hangup.

The *MASTER* sends *R*, *S* or *X* messages until all work for the remote system is complete, at which point an *H* message is sent. The *SLAVE* replies with *SY*, *SN*, *RY*, *RN*, *HY*, *HN*, *XY*, or *XN*, corresponding to *yes* or *no* for each request.

An *N* response can be followed by a number giving the reson for the failure:

N0
Copy failed (reason not given by remote system).

N1
Local access to file denied.

N2
Remote access to path or file denied.

N3
System error – bad *uucp* command generated.

N4
Remote system cannot create temporary file.

N5
Cannot copy to file or directory – file left in *pubdir/user/file*.

N6
Cannot copy to file or directory – file left in *pubdir/user/file*.

The send and receive replies are based on permission to access the requested file or directory using the *USERFILE* and read/write permissions of the file or directory.

After each file is copied into the spool directory of the receiving system, a copy-complete message is sent by the receiver of the file. The message *CY* is sent if the file has successfully been moved from the spool directory to the destination.

If the file was not successfully moved from the spool directory to the destination, a *CN* message is sent, the file is put in the public directory (usually */usr/spool/uucppublic*), and the requester is notified by mail.

The requests and results are logged on both systems.

The hangup response is determined by a work scan of the *SLAVE*'s spool directory. If work for the remote system exists an *HN* message is sent and the programs switch roles. If no work exists, an *HY* response is sent.

Conversation Termination

When the *MASTER* receives a *HY* message, the *MASTER* echoes the message back to the *SLAVE* and the protocols are turned off. Each program sends a final "OO" (Over and Out) message to the other. The original *SLAVE* program cleans up and terminate. The *MASTER* then proceeds to call other systems unless a "–s" option was specified.

C.5. UULOG — UUCP Log Inquiry

When a *uucp* program can not make a log entry directly into the *LOGFILE* an individual log file is created: a file with prefix *LOG*. This sometimes occurs when more than one *uucp* process is running. Periodically, *uulog* may be executed to append these files to the *LOGFILE*.

The *uulog* program may also be used to request the output of *LOGFILE* entries. The request is specified by the use of the options:

–s*sys*	Print entries where *sys* is the remote system name,
–u*user*	Print entries for user *user*.

The intersection of lines satisfying the two options is output. A null *sys* or *user* means all system names or users respectively.

Debugging options available are:

–x*num*	*Num* is a level number between 1 and 9; higher numbers give more debug output.
–n*secs*	Time out lock on log file if older than *secs* seconds.

C.6. UUCLEAN — UUCP Spool Directory Cleanup

Uuclean is typically started by the *uucp* daily daemon. Its function is to remove files from the spool directory that are more than three days old. These are usually files for work that can not be completed. The requester of this work is notified that the files have been deleted.

There are several options:

–d*dir*	The directory to be scanned is *dir*.
–m	Send mail to the owner of each file being removed. Note that most files put into the spool directory are owned by the owner of the *uucp* programs since the setuid bit will be set on these programs. This mail is sometimes useful for administration.
–n*hours*	Change the aging time from 72 hours to *hours* hours.

–p*pre* Examine files with prefix *pre* for deletion. Up to 10 of these options may be specified.

–x*num* This is the level of debugging output desired.

C.7. Security

CAUTION **The *uucp* system, left unrestricted, will let any outside user execute any commands and copy out/in any file that is readable/writable by a *uucp* login user. It is up to the individual sites to be aware of this and apply the protections that they feel are necessary.**

There are several security features available aside from the normal file mode protections. These must be set up by the administrator of the *uucp* system.

- The login for *uucp* does not get a standard shell. Instead, the *uucico* program is started so that all work is done through *uucico*.
- The owner of the *uucp* programs should be an administrative login. It should not be one of the logins used for remote system access to *uucp*.
- All *uucp* logins should have passwords.
- A path check is done on file names that are to be sent or received. The *USERFILE* supplies the information for these checks. The *USERFILE* can also be set up to require call-back for certain login-ids. See the "Files Required For Execution" section for the file description.
- A conversation sequence count can be set up so that the called system can be more confident of the caller's identity.
- The *uuxqt* program reads a file containing a list of commands that it will execute. A "PATH" shell statement is prepended to the command line as specified in the *uuxqt* program. The installer may modify the list or remove the restrictions as desired.
- The *L.sys* file should be owned by the *uucp* administrative login and have mode 0400 to protect the phone numbers and login information for remote sites.
- The programs *uucp*, *uucico*, *uux*, *uuxqt*, *uulog*, and *uuclean* should be owned by the *uucp* administrative login, have the setuid bit set, and have only execute permissions.

C.8. UUCP Installation

It is assumed that the *login name* used by a remote computer to call into a local computer is not the same as the login name of a normal user or the *uucp* administrative login. However, several remote computers may use the same login name. It is suggested that the installer follow the convention of using the letter "U" followed by the system name as the login name for each system. For example, use login name *Uusg* for the *usg* system.

Each computer should be given a unique *system name* that is transmitted at the start of each call. This name identifies the calling machine to the called machine. The *login/system* names are used for security as described later in the *USERFILE*

section.

There are several source modifications that may be required before the system programs are compiled. These relate to the directories, local system name, and attributes of the local environment.

The *uucp* system uses several directories:

sources	(/usr/src/cmd/uucp) – This directory contains the *uucp* system source files.
program	(/usr/lib/uucp) – This is the directory used for some of the executable system programs and the system files. Some of the programs reside in */usr/bin*.
spool	(/usr/spool/uucp) – This is the *uucp* system spool directory.
xqtdir	(/usr/lib/uucp/.XQTDIR) – This directory is used during execution of the *uux* scripts.

The names in parentheses above are the default values for the directories. The italicized names *sources, program, xqtdir,* and *spool* are used in the following text to represent the appropriate directory names.

There are two files that may require modification, namely the *Makefile* file and the *uucp.h* file. In addition, the *uuxqt.c* program may be modified as indicated in the section entitled *Security*, above. The following sections describe the modifications.

uucp.h Modification

Several manifests in *uucp.h* may need modification for the local system environment:

UNAME	should be defined if the ''uname'' function is available.
ACULAST	is the character required by the ACU as the last character. For most systems, it is a ''–''. This is only relevant if you are using a DN11 autodialler.
DIALOUT	should be defined if the C library routine ''dialout'' is available.
UUDIR	should be defined if the *uucp* subdirectory *syskludge* is being used.
UUNAME	should be defined if the system name should be read from the SYSTEMNAME file.
UUSTAT	should be defined if you need the *uustat* program.
UUSUB	should be defined if you need the *uusub* program.

Makefile Modification

There are several *make* variable definitions that may need modification:

INSDIR	is the *program* directory (for example, INSDIR=/usr/lib/uucp). This parameter is used if ''make cp'' or ''make install'' is used.

PUBDIR	is a public directory for remote access. This is also the login directory for remote *uucp* users. It should be the same as that defined in *uucp.h*.
SPOOL	is the uucp spool directory. This should be the same as that defined in *uucp.h*.
XQTDIR	is the directory for uuxqt to use for command execution. It is also defined in *uucp.h*.
OWNER	is the administrative login for *uucp*.
LIBS	should include *syskludge/syskludge.a if the syskludge library is used. UUDIR should be defined in uucp.h*.
CFLAGS	add –DVMUNIX if on a VMUNIX system.

Compiling the System

The command:

```
make install
```

makes the required directories, compiles all programs, sets the proper file modes, and copies the programs to the proper directories. This command should be run as *root*. The command:

```
make
```

compiles the entire system.

The programs *uucp*, *uux*, and *uulog* should be put in */usr/bin*. The programs *uuxqt*, *uucico*, and *uuclean* should be put in the *program* directory.

Files Required for Execution

Six files are required for execution. They should reside in the *program* directory. The field separator for all files is a space. The required files are summarized here, and the following subsections describe them in detail.

L-devices	Contains call unit information.
L-dialcodes	Contains dialcode abbreviations.
USERFILE	Contains user accessibility and constraint information.
L.sys	Contains information about the systems which local *uucp* programs can call.
L.cmds	Contains commands which *uuxqt* is allowed to execute.
SYSTEMNAME	Contains the name of the system.

L-devices — Call Unit Information File

L-devices contains call-unit device and hardwired connection information. The special device files are assumed to be in the */dev* directory. The format for each entry in the *L-devices* file is:

type line call-unit speed

type is a device type such as ACU or DIR. The currently supported device types are described later. The field can also be used to specify particular ACUs for some calls by using a suffix on the ACU field (ACUDF03wats, for instance). This name should be used in *L.sys*.

line is the device for the line (for example, cul0 if using a DN11, otherwise it is the same as the call-unit field).

call-unit is the automatic call unit associated with *line* (for example, cua0). Hardwired lines have a number "0" in this field.

speed is the line speed.

For example, an entry in the *L-devices* file like this:

```
ACU  cul0  cua0  300
```

would be set up for a system that has a DN11 device "/dev/cul0" wired to a call-unit "/dev/cua0" for use at 300 baud. An entry like:

```
ACUDF03  cua0  cua0  1200
```

would be set up for a system that has a DF03 device "/dev/cua0" for use at 1200 baud.

L-dialcodes — Dial Code Abbreviations File

L-dialcodes contains the dialcode abbreviations used in the *L.sys* file (for example, py, mh, boston). The entry format is:

```
abb  dial-seq
```

abb is the abbreviation,

dial-seq is the dial sequence to call that location.

For example, a line in the *L-dialcodes* file that looks like:

```
py  165-
```

would be set up so that entry py7777 would send 165–7777 to the dial-unit.

USERFILE

USERFILE contains user accessibility information. It specifies four types of constraint:

[1] which files can be accessed by a normal user of the local machine,

[2] which files can be accessed from a remote computer,

[3] which login name is used by a particular remote computer,

[4] whether a remote computer should be called back in order to confirm its identity.

Each line in *USERFILE* has the format:

login,sys [c] path-name [path-name] ...

login is the login name for a user or the remote computer,

sys is the system name for a remote computer,

c is the optional *call-back required* flag,

path-name is a path-name prefix that is acceptable for *sys*.

The constraints are implemented as follows.

[1] When the program is obeying a command stored on the local machine (*MASTER* mode) the path-names allowed are those given on the first line in the *USERFILE* that has the login name of the user who entered the command. If no such line is found, the first line with a *null* login name is used.

[2] When the program is responding to a command from a remote machine (*SLAVE* mode) the path-names allowed are those given on the first line in the file that has the system name that matches the remote machine. If no such line is found, the first one with a *null* system name is used.

[3] When a remote computer logs in, the login name that it uses *must* appear in the *USERFILE*. There may be several lines with the same login name but one of them must either have the name of the remote system or must contain a *null* system name.

[4] If the line matched in ([3]) contains a ''c'', the remote machine is called back before any transactions take place.

Examples

The line:

```
u,m  /usr/xyz
```

allows machine *m* to login with name *u* and request the transfer of files whose names start with */usr/xyz*.

The line:

```
dan,  /usr/dan
```

allows the ordinary user *dan* to issue commands for files whose name starts with */usr/dan*. (Note that this type of restriction is seldom used.)

The lines:

```
u,m /usr/xyz  /usr/spool
u,  /usr/spool
```

allows any remote machine to login with name *u*. If its system name is not *m*, it can only ask to transfer files whose names start with */usr/spool*. If it is system *m*, it can send files from path */usr/xyz* as well as */usr/spool*.

The lines:

```
root,  /
,   /usr
```

allow any user to transfer files beginning with */usr*, but the user with login *root* can transfer any file. Note that any file that is to be transferred must be readable by anybody. The *USERFILE* is normally set up as follows:

```
,myname /
, /usr/spool/uucppublic
```

where *myname* is the name of the current system. These lines allow any user on the current machine to access any file (subject to the normal permissions on the file) for *uucp* transfer, whereas users on other machines can only access files in */usr/spool/uucppublic*.

L.sys

Each entry in *L.sys* represents one system that can be called by the local *uucp* programs. More than one line may be present for a particular system. In this case, the additional lines represent alternative communication paths that are tried in sequential order. The fields are described below.

system name
: The name of the remote system.

time
: This is a string that indicates the days-of-week and times-of-day when the system should be called (for example, MoTuTh0800–1730).

  The day portion may be a list containing some of

  Su Mo Tu We Th Fr Sa

  or it may be *Wk* for any week-day or *Any* for any day.

  The time should be a range of times (for example, 0800–1230). If no time portion is specified, any time of day is assumed to be okay for the call. Note that a time range that spans 0000 is permitted, for example, 0800-0600 means all times are ok other than times between 6 and 8 am.

  A time specification of ''None'' is often used for a passive system, that is, one which cannot call the specified system. In this case the following fields may be omitted. Note that the string ''None'' has no special significance, but is merely a string that is not a correct time specification.

  Several day-time specifications may be present, separated by a | character.

  An optional subfield is available to indicate the minimum time (minutes) before a retry following a failed attempt. The subfield separator is a '','. For example, Any,9 means call any time but don't allow another call until at least 9 minutes after a failure has occurred.

device — This fields either starts with *ACU*, or is the hardwired device to be used for the call. For the hardwired case, the last part of the special file name is used (tty0, for instance).

class — This is usually the line speed for the call (for example, 300). The exception is when the C library routine ''dialout'' is available in which case this is the dialout class.

phone — The phone number is made up of an optional alphabetic abbreviation and a numeric part. The abbreviation should be one that appears in the *L-dialcodes* file (for example, mh5900, boston995–9980). For the hardwired devices, this field contains the same string as used for the *device* field.

login — The login information is given as a series of fields and subfields in the format

```
[ expect  send ]  . . .
```

where *expect* is the string expected to be read and *send* is the string to be sent when the *expect* string is received.

The expect field may be made up of subfields of the form

```
expect[-send-expect] . . .
```

where the *send* is sent if the prior *expect* is *not* successfully read and the *expect* following the *send* is the next expected string. For example:

```
login--login
```

expects to see the word *login*; if it gets it, the program proceeds to the next field; if it does not get *login*, it sends *null* followed by a new line, then expects *login* again.

There are two special names available to be sent during the login sequence. The string *EOT* sends an EOT character and the string *BREAK* tries to send a BREAK character. The *BREAK* character is simulated using line speed changes and null characters and may not work on all devices and/or systems. A number from 1 to 9 may follow the *BREAK*. For example, *BREAK1* sends 1 null character instead of the default of 3. Note that *BREAK1* usually works best for 300/1200 baud lines.

The following escape sequences are also recognized:

\r — send a carriage-return.

\n — send a newline (linefeed) character.

\d — delay for 1 second.

\c suppress newline at end of *send* string.

\s send a space.

A typical entry in the L.sys file would be:

```
sys  Any  ACU  300  mh7654  login:-EOT-login:-BREAK-login:  uucp
```

The expect algorithm matches all or part of the input string as illustrated in the password field above. Very complex expect-send sequences are often required if the called system is connected to a terminal concentrator or a front end.

L.cmds

L.cmds contains a list of commands which *uuxqt* is allowed to execute. The commands should be one per line. At a minimum, *L.cmds* should contain the command "rmail". Other commands often allowed are "rnews", "uusend", and "lpr".

Device Types

The currently supported device types are:

Type Code	Device
ACU	DEC DN11
ACUDN11	DEC DN11
ACUDF02	DEC DF02 (300 baud only)
ACUDF03	DEC DF03 (1200 baud only)
ACUVENTEL	Ventel MD212 Plus
DIR	Hardwired Line.

The DN11 is only available on DEC PDP-11 and VAX systems. The DEC DF02, DF03, and Ventel MD212 Plus can be connected to any system using a standard RS232 terminal interface. When connecting one of these devices to a terminal line, the device node (/dev/tty*x*) should be renamed (to /dev/cua0 for instance) to emphasize that the line is for dialout only and to prevent accidentally starting a login process for that line.

The device type specified in the *L-devices* file should be one of those listed above, optionally qualified further by additional characters after the device type. The device type specified in the *L.sys* file should be a prefix of one of the devices specified in the *L-devices* file. For example, assume you have two DF03's, one connected to a local telephone line and the other connected to a WATS line. The *L-devices* file could be set up as follows:

```
ACUDF03local  cua0  cua0  1200
ACUDF03wats   cua0  cua0  1200
```

To call a system using only the WATS line, specify ACUDF03wats in the device type field. Similarly, to call a system using the local telephone line, specify ACUDF03local. To call a system using either DF03, specify ACUDF03 in the

L.sys file.

Note that the telephone numbers specified in the *L.sys* file will have a format dependent on the ACU device type. This is a deficiency which may be corrected in the future.

C.9. Administration

This section indicates some events and files that must be administered for the *uucp* system. Some administration can be accomplished by *shell files* initiated by *crontab* entries. Others may require manual intervention. Some sample *shell files* are given toward the end of this section.

SQFILE — Sequence Check File

SQFILE is set up in the *program* directory and contains an entry for each remote system with which you agree to perform conversation sequence checks. The initial entry is just the system name of the remote system. The first conversation adds the conversation count and the date/time of the most resent conversation. These items are updated with each conversation. If a sequence check fails, the entry will have to be adjusted manually. Note that this feature is rarely used.

TM — Temporary Data Files

These files are created in the *spool* directory while a file is being copied from a remote machine. Their names have the form

```
TM.pid.ddd
```

where *pid* is a process-id and *ddd* is a sequential three digit number starting at zero. After the entire file is received, the *TM* file is moved/copied to the requested destination. If processing is abnormally terminated the file remains in the spool directory. The leftover files should be periodically removed; the *uuclean* program is useful in this regard. The command

```
program/uuclean -pTM
```

removes all *TM* files older than three days.

LOG — Log Entry Files

During execution, log information is appended to the *LOGFILE*. If the *LOGFILE* is locked by another process, the log information is placed in individual log files with a with a *LOG* prefix. These individual files should be combined into the *LOGFILE* by using the *uulog* program. *Uulog* appends the contents of the individual log files onto the end of the *LOGFILE*. The command:

```
uulog
```

accomplishes the merge. Options are available to print some or all the log entries after the files are merged. The *LOGFILE* should be removed periodically.

The *LOG* files are created initially with mode 0222. If the program that creates the file terminates normally, it changes the mode to 0666. Aborted runs may leave the files with mode 0222 and the *uulog* program will not read or remove them. To remove them, either use *rm*, *uuclean*, or change the mode to 0666 and let *uulog* merge them into the *LOGFILE*.

STST — System Status Files

These files are created in the spool directory by the *uucico* program. They contain information such as login, dialup or sequence check failures or will contain a *TALKING* status when two machines are conversing. The form of the file name is

STST.`sys`

where *sys* is the remote system name, truncated to seven characters.

For ordinary failures, such as dialup or login, the file prevents repeated tries for about 55 minutes. This is the default time; it can be changed on an individual system basis by a subfield of the time field in the *L.sys* file. For sequence check failures, the file must be removed before any future attempts to converse with that remote system. Retries are accomplished by starting *uucico* from *crontab*, usually hourly.

LCK — Lock Files

Lock files are created for each device in use (for example, automatic calling unit) and each system conversing. This prevents duplicate conversations and multiple attempts to use the same device. The form of the lock file name is:

LCK..`str`

where *str* is either a device or system name. The files may be left in the spool directory if runs abort (usually only on system crashes). They are ignored (reused) after 1.5 hours. When runs abort and calls are desired before the time limit, the lock files should be removed.

Shell Files

The *uucp* program spools work and attempts to start the *uucico* program, but *uucico* will not always be able to execute the request immediately. Therefore, the *uucico* program should be periodically started. The command to start *uucico* can be put in a "shell" file with a command to merge *LOG.* files and started by a crontab entry on an hourly basis. The file could contain the commands:

```
/usr/bin/uulog
program/uucico   -r1  -sinter
program/uucico   -r1
```

The "–r1" option is required to start the *uucico* program in *MASTER* mode. The "–s" option can be used for polling as illustrated in the second line where machine *inter* is being polled. The third line processes all other spooled work.

Another shell file may be set up on a daily basis to remove *TM*, *ST* and *LCK* files and *C.* or *D.* files for work that can not be accomplished for reasons like bad phone number, login changes, and so on. A shell file containing commands like:

```
program/uuclean    -pTM -pC. -pD.
program/uuclean    -pST -pLCK -n12
```

can be used. Note that the "–n12" option causes the *ST* and *LCK* files older than 12 hours to be deleted. The absence of the "–n" option uses a three day time limit.

A daily or weekly shell should also be created to remove or save old *LOGFILE*'s. A shell like:

```
cp spool/LOGFILE    spool/o.LOGFILE
rm spool/LOGFILE
```

can be used.

The shell files in *program*/uucp.* do a more extensive job than that described here. They should be started by entries in *crontab*. Read the shell files for more information.

Login Entry

Two or more logins should be set up for *uucp*. One should be an administrative login: the owner of all the uucp programs, directories and files. All others are used by remote systems to access the uucp system. Each of the */etc/passwd* entries for the *access* logins should have *program*/uucico as the shell to be executed. The login directory should be the public directory (usually */usr/spool/uucppublic*) for both the administrative login and the access logins. The various *access* login names are used in the *USERFILE* to restrict file access.

File Modes

The programs *uucp*, *uux*, *uucico*, *uulog*, *uuclean*, and *uuxqt* should be owned by the *uucp* administrative login with the "setuid" bit set and only execute permissions (mode 04111). The *L.sys*, *SQFILE*, and the *USERFILE*, which are put in the *program* directory should be owned by the *uucp* administrative login and set with mode 0400. The mode of *spool* should be 0755. The mode of *xqtdir* should be 0777. The *L-dialcodes* and the *L-devices* files should have mode 0444.

D

Sendmail Installation and Operation

Sendmail Installation and Operation 329

D.1. Basic Installation 329
Off-the-Shelf Configurations 330
Installation Using the Makefile 330
Installation by Hand 330
D.2. Normal Operations 332
Quick Configuration Startup 332
The System Log 332
The Mail Queue 332
The Local Alias Database 335
Yellow Pages Aliases 336
Per-User Forwarding (.forward Files) 337
Special Header Lines 337
D.3. Arguments 337
Queue Interval 337
Daemon Mode 337
Forcing the Queue 338
Debugging 338
Trying a Different Configuration File 338
Changing the Values of Options 338
D.4. Tuning 338
Timeouts 339
Delivery Mode 339

Load Limiting 340
Log Level 340
File Modes 341
D.5. The Whole Scoop on the Configuration File 341
The Syntax 342
The Semantics 344
Building a Configuration File from Scratch 351
D.6. Command Line Flags 355
D.7. Configuration Options 357
D.8. Mailer Flags 359
D.9. Summary of Support Files 361

D

Sendmail Installation and Operation

`sendmail` implements a general purpose internetwork mail routing facility under the UNIX operating system. It is not tied to any one transport protocol – its function may be likened to a crossbar switch, relaying messages from one domain into another. In the process, it can do a limited amount of message header editing to put the message into a format that is appropriate for the receiving domain. All of this is done under the control of a configuration file.

Due to the requirements of flexibility for `sendmail`, the configuration file can seem somewhat unapproachable. However, there are only a few basic configurations for most sites, for which standard configuration files have been supplied. Most other configurations can be built by adjusting an existing configuration file incrementally.

Although `sendmail` is intended to run without the need for monitoring, it has a number of features that may be used to monitor or adjust the operation under unusual circumstances. These features are described.

Section D.1 describes how to do a basic sendmail installation. Section D.2 explains the day-to-day information you should know to maintain your mail system. If you have a relatively normal site, these two sections should contain sufficient information for you to install `sendmail` and keep it happy. Section D.4 describes some parameters that may be safely tweaked. Section D.3 has information regarding the command line arguments. Section D.5 contains the nitty-gritty information about the configuration file. This section is for masochists and people who must write their own configuration file. Sections D.6 through D.9 give a brief but detailed explanation of a number of features not described in the rest of the paper.

The references in this paper are actually found in the companion paper *Sendmail — An Internetwork Mail Router*. Read that paper before this one to gain a basic understanding of how the pieces fit together.

D.1. Basic Installation

There are two basic steps to installing `sendmail`. The first, and hardest, part is to build the configuration file. The configuration file describes the mailers it knows about, how to parse addresses, how to rewrite the message header, and the settings of various options. When `sendmail` starts up it reads the file. Although the configuration file is quite complex, a configuration can usually be built by adjusting an existing off-the-shelf configuration. The second step is actually doing the installation, which means creating the necessary files, etc.

The remainder of this section will describe the installation of `sendmail` assuming you can use one of the existing configurations and that the standard installation parameters are acceptable. For source distributions, all pathnames and examples are given from the root of the `sendmail` subtree.

Off-the-Shelf Configurations

You can produce your own *sendmail.cf* file from one of the generic configuration files provided with this release: */usr/lib/sendmail.main.cf* or */usr/lib/sendmail.subsidary.cf*. Procedures are given in *Setting Up the Mail Routing System* in the *Communications* chapter of this manual.

Installation Using the Makefile

A makefile exists in the root of the `sendmail` directory that will do all of these steps for the standard Sun system. It may have to be slightly tailored for use on other systems.

Before using this makefile, you should already have created your configuration file and left it in the file *cf/system_name.cf* where *system_name* is the name of your system (that is, what is returned by *hostname* (1)). You should also examine the file *md/config.m4* and change the *m4* macros there to reflect any libraries and compilation flags you may need.

The basic installation procedure is to type:

```
% make
% make install
```

in the source directory of `sendmail`. This will make all binaries and install them in the standard places. The second *make* command must be executed as the superuser (root).

Installation by Hand

Along with building a configuration file, you will have to install the `sendmail` startup into your UNIX system. If you are doing this installation in conjunction with a regular Sun UNIX install, these steps will already be complete. Many of these steps will have to be executed as the superuser (root).

/usr/lib/sendmail

The binary for `sendmail` is located in */usr/lib*.

/usr/lib/sendmail.cf

The configuration file that you created earlier should be installed in */usr/lib/sendmail.cf*; see the *Setting up the Mail System* section in the *System Set-up and Operation* chapter.

/usr/spool/mqueue

The directory */usr/spool/mqueue* should be created to hold the mail queue. This directory should be mode 777 unless `sendmail` is run setuid, when *mqueue* should be owned by the `sendmail` owner and mode 755.

/usr/lib/aliases*

The file */usr/lib/aliases* is the master file for local aliases. On *nd* servers and clients, this file is a symbolic link to */private/aliases*. It can be omitted if only domain-wide aliases are used.

/etc/rc

It will be necessary to start up the `sendmail` daemon when your system reboots. This daemon performs two functions: it listens on the SMTP socket for connections (to receive mail from a remote system) and it processes the queue periodically to insure that mail gets delivered when hosts come up.

The following lines should be in */etc/rc* (or */etc/rc.local* as appropriate) in the area where it is starting up the daemons:

```
if [ -f /usr/lib/sendmail ]; then
    (cd /usr/spool/mqueue; rm -f [lnx]f*)
    /usr/lib/sendmail -bd -q30m &
    echo -n ' sendmail' >/dev/console
fi
```

The *cd* and *rm* commands insure that all lock files have been removed; extraneous lock files may be left around if the system goes down in the middle of processing a message. The line that actually invokes `sendmail` has two flags: `-bd` causes it to listen on the SMTP port, and `-q30m` causes it to run the queue every half hour.

/usr/lib/sendmail.fc

If you intend to install the frozen version of the configuration file (for quick startup) you should create the file */usr/lib/sendmail.fc* and initialize it. This step may be safely skipped.

```
% cp /dev/null /usr/lib/sendmail.fc
% /usr/lib/sendmail -bz
```

/usr/lib/sendmail.hf

This is the help file used by the SMTP **HELP** command. The file is already installed in the distribution.

/usr/lib/sendmail.st

If you wish to collect statistics about your mail traffic, create the file */usr/lib/sendmail.st*:

```
% cp /dev/null /usr/lib/sendmail.st
% chmod 666 /usr/lib/sendmail.st
```

This file does not grow. It is printed with the program *aux/mailstats*.

/usr/etc/in.syslog

You may want to run the *syslog* program on one host in your network (to collect log information about `sendmail`). This program normally resides in */usr/etc/in.syslog*, with support files */etc/syslog.conf* and */etc/syslog.pid*. The file */etc/syslog.conf* describes the file(s) that `sendmail` will log in. For a complete description of *syslog*, see the manual page for *syslog*(8).

D.2. Normal Operations

Quick Configuration Startup

A fast version of the configuration file may be set up by using the `-bz` flag:

```
% /usr/lib/sendmail -bz
```

This creates the file */usr/lib/sendmail.fc* (''frozen configuration''). This file is an image of `sendmail`'s data space after reading in the configuration file. If this file exists, it is used instead of */usr/lib/sendmail.cf*. *sendmail.fc* must be rebuilt manually every time `sendmail.cf` is changed.

The frozen configuration file will be ignored if a `-C` flag is specified or if `sendmail` detects that it is out of date. However, the heuristics are not strong so this should not be trusted.

The System Log

The system log is supported by the *syslog* (8) program.

Format

Each line in the system log consists of a timestamp, the name of the machine that generated it (for logging from several machines over the ethernet), the word ''sendmail:'', and a message.

Levels

If you have *syslog* (8) or an equivalent installed, you will be able to do logging. *Syslog* installation is performed by the *setup* program during first-time UNIX installation for Sun systems. There is a large amount of information that can be logged. The log is arranged as a succession of levels. At the lowest level only extremely strange situations are logged. At the highest level, even the most mundane and uninteresting events are recorded for posterity. As a convention, log levels under ten are considered ''useful;'' log levels above ten are usually for debugging purposes.

A complete description of the log levels is given in section 4.3.

The Mail Queue

The mail queue should be processed transparently. However, you may find that manual intervention is sometimes necessary. For example, if a major host is down for a period of time the queue may become clogged. Although `sendmail` ought to recover gracefully when the host comes up, you may find performance unacceptably bad in the meantime.

Printing the Queue

The contents of the queue can be printed by specifying the `-bp` flag to `sendmail`:

```
% /usr/lib/sendmail -bp
```

This will produce a listing of the queue id's, the size of the message, the date the message entered the queue, and the sender and recipients.

Format of Queue Files

All queue files have the form *xfAA99999* where *AA99999* is the *id* for this file and the *x* is a type. The types are:

d The data file. The message body (excluding the header) is kept in this file.

l The lock file. If this file exists, the job is currently being processed, and a queue run will not process the file. For that reason, an extraneous *lf* file can cause a job to apparently disappear (it will not even time out!).

n This file is created when an id is being created. It is a separate file to insure that no mail can ever be destroyed due to a race condition. It should exist for no more than a few milliseconds at any given time.

q The queue control file. This file contains the information necessary to process the job.

t A temporary file. These are an image of the *qf* file when it is being rebuilt. It should be renamed to a *qf* file very quickly.

x A transcript file, existing during the life of a session showing everything that happens during that session.

The *qf* file is structured as a series of lines each beginning with a code letter. The lines are as follows:

D The name of the data file. There may only be one of these lines.

H A header definition. There may be any number of these lines. The order is important: they represent the order in the final message. These use the same syntax as header definitions in the configuration file.

R A recipient address. This will normally be completely aliased, but is actually realiased when the job is processed. There will be one line for each recipient.

S The sender address. There may only be one of these lines.

T The job creation time. This is used to compute when to time out the job.

P The current message priority. This is used to order the queue. Higher numbers mean lower priorities. The priority increases as the message sits in the queue. The initial priority depends on the message class and the size of the message.

M A message. This line is printed by using `sendmail` with the **–bp** flag, and is generally used to store status information. It can contain any text.

As an example, the following is a queue file sent to ''mckusick@calder'' and ''wnj :''

```
DdfA13557
Seric
T404261372
P132
H?D?date: 23-Oct-82 15:49:32-PDT (Sat)
H?F?from: eric (Eric Allman)
H?x?full-name: Eric Allman
Hsubject: this is an example message
Hmessage-id: <8209232249.13557@UCBARPA.BERKELEY.ARPA>
Hreceived: by UCBARPA.BERKELEY.ARPA (3.227 [10/22/82])
    id A13557; 23-Oct-82 15:49:32-PDT (Sat)
Hphone: (415) 548-3211
HTo: mckusick@calder, wnj
Rmckusick@calder
Rwnj
```

This shows the name of the data file, the person who sent the message, the submission time (in seconds since January 1, 1970), the message priority, the message class, the recipients, and the headers for the message.

Forcing the Queue

`sendmail` should run the queue automatically at intervals. The algorithm is to read and sort the queue, and then to attempt to process all jobs in order. When it attempts to run the job, `sendmail` first checks to see if the job is locked. If so, it ignores the job.

There is no attempt to insure that only one queue processor exists at any time, since there is no guarantee that a job cannot take forever to process. Due to the locking algorithm, it is impossible for one job to freeze the queue. However, an uncooperative recipient host or a program recipient that never returns can accumulate many processes in your system. Unfortunately, there is no way to resolve this without violating the protocol.

In some cases, you may find that a major host going down for a couple of days may create a prohibitively large queue. This will result in `sendmail` spending an inordinate amount of time sorting the queue. This situation can be fixed by moving the queue to a temporary place and creating a new queue. The old queue can be run later when the offending host returns to service.

To do this, it is acceptable to move the entire queue directory:

```
% cd /usr/spool
% mv mqueue omqueue; mkdir mqueue; chmod 777 mqueue
```

You should then kill the existing daemon (since it will still be processing in the old queue directory) and create a new daemon.

To run the old mail queue, run the following command:

```
% /usr/lib/sendmail -oQ/usr/spool/omqueue -q
```

The `-oQ` flag specifies an alternate queue directory and the `-q` flag says to just run every job in the queue. If you have a tendency toward voyeurism, you can use the `-v` flag to watch what is going on.

When the queue is finally emptied, you can remove the directory:

```
% rmdir /usr/spool/omqueue
```

You can also run a subset of the queue at any time with the `-R` *string* (run queue where any recipient address matches *string*) or with `-M` *nnnnn* (run just one message, with queue id *nnnnn*).

The Local Alias Database

The local alias database exists in two forms. (See the section below for information about domain-wide aliases with the yellow pages.) One is a text form, maintained in the file */usr/lib/aliases*. The aliases are of the form

```
name: name1, name2, ...
```

Only local names may be aliased; for example,

```
eric@mit-xx: eric@berkeley
```

will not have the desired effect. Aliases may be continued by starting any continuation lines with a space or a tab. Blank lines and lines beginning with a sharp sign (''#'') are comments.

The second form is processed by the *dbm*(3) library. This form is in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag*. This is the form that `sendmail` actually uses to resolve aliases. This technique is used to improve performance.

Rebuilding the Alias Database

The DBM version of the database may be rebuilt explicitly by executing the command

```
% newaliases
```

This is equivalent to giving `sendmail` the `-bi` flag:

```
% /usr/lib/sendmail -bi
```

If the **D** option is specified in the configuration, `sendmail` will rebuild the alias database automatically if possible when it is out of date. The conditions under which it will do this are:

1. The DBM version of the database is mode 666. --or--
2. `sendmail` is running setuid to root.

Auto-rebuild can be dangerous on heavily loaded machines with large alias files; if it might take more than five minutes to rebuild the database, there is a chance that several processes will start the rebuild process simultaneously.

Potential Problems

There are a number of problems that can occur with the alias database. They all result from a `sendmail` process accessing the DBM version while it is only partially built. This can happen under two circumstances: One process accesses the database while another process is rebuilding it, or the process rebuilding the database dies (due to being killed or a system crash) before completing the rebuild.

`sendmail` has two techniques to try to relieve these problems. First, it ignores interrupts while rebuilding the database; this avoids the problem of someone

aborting the process leaving a partially rebuilt database. Second, at the end of the rebuild it adds an alias of the form

```
@: @
```

(which is not normally legal). Before `sendmail` will access the database, it checks to insure that this entry exists. The **a** option is required in the configuration for this action to occur. It will wait for the time specified in the **a** option for this entry to appear, at which point it will force a rebuild itself.[8]

List Owners

If an error occurs on sending to a certain address, say *x*, `sendmail` will look for an alias of the form "owner-*x*" to receive the errors. This is typically useful for a mailing list where the submitter of the list has no control over the maintanence of the list itself; in this case the list maintainer would be the owner of the list. For example:

```
unix-wizards: eric@ucbarpa, wnj@monet, nosuchuser,
    sam@matisse
owner-unix-wizards: eric@ucbarpa
```

would cause "eric@ucbarpa" to get the error that will occur when someone sends to unix-wizards due to the inclusion of "nosuchuser" on the list.

Yellow Pages Aliases

Mail aliases can also be obtained from a domain-wide database using the yellow pages. The yellow pages database is usually the `/usr/lib/aliases` file from some central mail machine for the group or department. This database is searched after local aliases but before `.forward` files. The yellow pages map name can be specified with the `Y` configuration option; the default is `mail.alaises`. This feature can be disabled by turning off yellow pages, by not having the given map in the default domain, or by including a `Y` option with a null string as its argument.

Naming Conventions

It is a good idea to follow a standard convention for naming users. For example, at Sun every user is given an alias consisting of their first initial followed by last name. These aliases exist on the machine that relays mail to the outside world, and are also used as the domain-wide `mail.aliases` map. This way, on any machine that uses domain-wide yellow pages aliases, you can just mail to the first initial and last name instead of having to remember the specific user and host name.

Potential Problems

Extra care must be taken to avoid loops and inconsistent databases when both local and domain-wide aliases are used. For example, consider a user named 'smith' who moves from a machine 'a' to a machine named 'b'. One might be tempted to do this by adding a local alias on machine 'a' that forwards 'smith@a' to 'smith@b', but if 'smith' maps to 'smith@a' in the domain-wide database, a loop will result.

Another problem is with aliases that use `:include:` files. These aliases are forwarded to the machine with the name of the domain if they are encountered in

[8] Note: the **D** option must be specified in the configuration file for this operation to occur.

domain-wide aliases. Similarly, domain-wide aliases that do not resolve to an address with a host name are forwarded to the domain-name host. For these reasons, it is a good idea to have the domain name be a synonym for a host that handles mail for the domain.

Per-User Forwarding (.forward Files)

As an alternative to the alias database, any user may put a file with the name *.forward* in his or her home directory. If this file exists, `sendmail` redirects mail for that user to the list of addresses listed in the *.forward* file. For example, if the home directory for user "mckusick" has a *.forward* file with contents:

```
mckusick@ernie
kirk@calder
```

then any mail arriving for "mckusick" will be redirected to the specified accounts.

Special Header Lines

Several header lines have special interpretations defined by the configuration file. Others have interpretations built into `sendmail` that cannot be changed without changing the code. These builtins are described here.

Return-Receipt-To:

If this header is sent, a message will be sent to any specified addresses when the final delivery is complete, if the mailer has the **l** flag (local delivery) set in the mailer descriptor.

Errors-To:

If errors occur anywhere during processing, this header will cause error messages to go to the listed addresses rather than to the sender. This is intended for mailing lists.

To:

If a message comes in with no recipients listed in the message (in a To:, Cc:, or Bcc: line) then `sendmail` will add a "To:" header line for each recipient specified on the `sendmail` command line.

At least one recipient line is required under RFC 822.

D.3. Arguments

The complete list of arguments to `sendmail` is described in detail in section D.6. Some important arguments are described here.

Queue Interval

`sendmail`" The amount of time between forking a process to run through the queue is defined by the `-q` flag. If you run in mode **f** or **a** this can be relatively large, since it will only be relevant when a host that was down comes back up. If you run in **q** mode it should be relatively short, since it defines the maximum amount of time that a message may sit in the queue.

Daemon Mode

If you allow incoming mail over an IPC connection, you should have a daemon running. This should be set by your */etc/rc* file using the `-bd` flag. The `-bd` flag and the `-q` flag may be combined in one call:

```
% /usr/lib/sendmail -bd -q30m
```

Forcing the Queue

In some cases you may find that the queue has gotten clogged for some reason. You can force a queue run using the `-q` flag (with no value). It is entertaining to use the `-v` flag (verbose) when this is done to watch what happens:

```
% /usr/lib/sendmail -q -v
```

Debugging

There are a fairly large number of debug flags built into `sendmail`. Each debug flag has a number and a level, where higher levels means to print out more information. The convention is that levels greater than nine are absurd, because they print out so much information that you wouldn't normally want to see them except for debugging that particular piece of code. Debug flags are set using the `-d` option; the syntax is:

```
debug-flag:   -d debug-list
debug-list:   debug-option [ , debug-option ]
debug-option: debug-range [ . debug-level ]
debug-range:  integer | integer - integer
debug-level:  integer
```

where spaces are for reading ease only. For example,

```
-d12     Set flag 12 to level 1
-d12.3   Set flag 12 to level 3
-d3-17   Set flags 3 through 17 to level 1
-d3-17.4     Set flags 3 through 17 to level 4
```

For a complete list of the available debug flags you will have to look at the code (they are too dynamic to keep this documentation up to date).

Trying a Different Configuration File

An alternative configuration file can be specified using the `-C` flag; for example,

```
% /usr/lib/sendmail -Ctest.cf
```

uses the configuration file *test.cf* instead of the default */usr/lib/sendmail.cf*. If the `-C` flag has no value it defaults to `sendmail.cf` in the current directory.

Changing the Values of Options

Options can be overridden using the `-o` flag. For example,

```
% /usr/lib/sendmail -oT2m
```

sets the **T** (timeout) option to two minutes for this run only.

D.4. Tuning

There are a number of configuration parameters you may want to change, depending on the requirements of your site. Most of these are set using an option in the configuration file. For example, the line "OT3d" sets option **T** to the value "3d" (three days).

Timeouts

All time intervals are set using a scaled syntax. For example, ''10m'' represents ten minutes, whereas ''2h30m'' represents two and a half hours. The full set of scales is:

```
s   seconds
m   minutes
h   hours
d   days
w   weeks
```

Queue Interval

The argument to the -q flag specifies how often a subdaemon will run the queue. This is typically set to between fifteen minutes and one hour.

Read Timeouts

It is possible to time out when reading the standard input or when reading from a remote SMTP server. Technically, this is not acceptable within the published protocols. However, it might be appropriate to set it to something large in certain environments (such as an hour). This will reduce the chance of large numbers of idle daemons piling up on your system. This timeout is set using the **r** option in the configuration file.

Message Timeouts

After sitting in the queue for a few days, a message will time out. This is to insure that at least the sender is aware of the inability to send a message. The timeout is typically set to three days. This timeout is set using the **T** option in the configuration file.

The time of submission is set in the queue, rather than the amount of time left until timeout. As a result, you can flush messages that have been hanging for a short period by running the queue with a short message timeout. For example,

```
% /usr/lib/sendmail -oT1d -q
```

will run the queue and flush anything that is one day old.

Delivery Mode

There are a number of delivery modes that `sendmail` can operate in, set by the **d** configuration option. These modes specify how quickly mail will be delivered. Legal modes are:

```
i   deliver interactively (synchronously)
b   deliver in background (asynchronously)
q   queue only (don't deliver)
```

There are tradeoffs. Mode ''i'' passes the maximum amount of information to the sender, but is hardly ever necessary. Mode ''q'' puts the minimum load on your machine, but means that delivery may be delayed for up to the queue interval. Mode ''b'' is probably a good compromise. However, this mode can cause large numbers of processes if you have a mailer that takes a long time to deliver a message.

Load Limiting

Often central mail machines can be overloaded. Of course the best solution is to dedicate a more powerful machine to handling mail, but load almost always will expand to consume whatever resources are allocated.

The **x** and **X** options can be used to limit the load caused by `sendmail`. Both of these configuration options take an argument that is an integer load average. For example, if you specify **x4** and **X8** then the **x** load limiting will be used when the load is above four, and the **X** load limiting will be used when the load is above eight. When the load is above the value specified in the **X** option, the SMTP server will not accept connections from the network (locally originiated mail and other mail such as UUCP are not affected). The **x** option has a more subtle effect, controlling whether messages are queued for later delivery or delivered immediately. The general idea is to always deliver 'small' messages immediately, and defer 'large' messages for delivery during off-peak periods.

The **q** option specifies the maximum size of message that will be delivered immediately. The 'size' of the message includes not only the number of bytes in the message, but also penalties for large number of recipients, and for delivery attempts that were unsucessful. The penalty per recipient is the compile-time parameter `WKRECIPFACT`, by default set to 50. The penalty per delivery attempt is the compile-time parameter `WKTIMEFACT`, by default set to 1000. The size limit is also dependent on current load, so that more and more messages are queued as load goes higher. If the load is one above the **x** threshold, then the limit is halved, if the load is two above the threshold, the limit is divided by three, etc. Note that this limit also applies to messages being delivered when running the queue, in contrast to earlier versions of `sendmail`.

The goal of load limiting is to prevent wasting time during loaded periods by attempting to deliver large messages, messages to many recipients, and messages to sites that have been down for a long time.

Log Level

The level of logging can be set for `sendmail`. The default using a standard configuration table is level 9. The levels are as follows:

0 No logging.

1 Major problems only.

2 Message collections and failed deliveries.

3 Successful deliveries.

4 Messages being defered (due to a host being down, etc.).

5 Normal message queueups.

6 Unusual but benign incidents, for example, trying to process a locked queue file.

9 Log internal queue id to external message id mappings. This can be useful for tracing a message as it travels between several hosts.

12 Several messages that are basically only of interest when debugging.

16 Verbose information regarding the queue.

File Modes

There are a number of files that may have a number of modes. The modes depend on what functionality you want and the level of security you require.

To Suid or not to Suid?

`sendmail` can safely be made setuid to root. At the point where it is about to *exec* (2) a mailer, it checks to see if the userid is zero; if so, it resets the userid and groupid to a default (set by the **u** and **g** options). (This can be overridden by setting the **S** flag to the mailer for mailers that are trusted and must be called as root.) However, this will cause mail processing to be accounted (using *sa* (8)) to root rather than to the user sending the mail.

Temporary File Modes

The mode of all temporary files that `sendmail` creates is determined by the **F** option. Reasonable values for this option are 0600 and 0644. If the more permissive mode is selected, it will not be necessary to run `sendmail` as root at all (even when running the queue), but will allow users to read mail in the queue.

Should my Alias Database be Writable?

At Sun Microsystems, we provide the alias database (*/usr/lib/aliases**) with mode 666. There are some dangers inherent in this approach: any user can add him-/her-self to any list, or can cause any user's mail to beforwarded elsewhere. However, we have found users to be basically trustworthy, and the cost of having a read-only database greater than the expense of finding and eradicating the rare nasty person.

D.5. The Whole Scoop on the Configuration File

This section describes the configuration file in detail, including hints on how to write one of your own if you have to.

There is one point that should be made clear immediately: the syntax of the configuration file is designed to be reasonably easy to parse, since parsing can be done every time `sendmail` starts up, rather than easy for a human to read or write.

`sendmail` uses single letters for several different functions:

- Command-line flags, see section D.6.
- Configuration options, see section D.7.
- Queue file line types, see the section above, *Format Of Queue Files*.
- Configuration file line types, see the section below, *The Syntax*.
- Mailer field names, see the section below, *M Define Mailer*.
- Mailer flags, see section D.8.
- Macro names, see the section below, *Special Macros, Conditionals*.
- Class names, see the configuration file.

An overview of the configuration file is given first, followed by details of the semantics.

The Syntax

The configuration file is organized as a series of lines, each of which begins with a single character defining the semantics for the rest of the line. Lines beginning with a space or a tab are continuation lines (although the semantics are not well defined in many places). Blank lines and lines beginning with a sharp symbol ('#') are comments.

R and S — Rewriting Rules

The core of address parsing are the rewriting rules. These are an ordered production system. `sendmail` scans through the set of rewriting rules looking for a match on the left hand side (LHS) of the rule. When a rule matches, the address is replaced by the right hand side (RHS) of the rule.

There are several sets of rewriting rules. Some of the rewriting sets are used internally and must have specific semantics. Other rewriting sets do not have specifically assigned semantics, and may be referenced by the mailer definitions or by other rewriting sets.

The syntax of these two commands are:

S*n*

Sets the current ruleset being collected to *n*. If you begin a ruleset more than once it deletes the old definition.

R*lhs rhs comments*

The fields must be separated by at least one tab character; there may be embedded spaces in the fields. The *lhs* is a pattern that is applied to the input. If it matches, the input is rewritten to the *rhs*. The *comments* are ignored.

D — Define Macro

Macros are named with a single character. These may be selected from the entire ASCII set, but user-defined macros should be selected from the set of upper case letters only. Lower case letters and special symbols are used internally.

The syntax for macro definitions is:

D*xval*

where *x* is the name of the macro and *val* is the value it should have. Macros can be interpolated in most places using the escape sequence **$***x*.

C and F — Define Classes

Classes of words may be defined to match on the left hand side of rewriting rules. For example a class of all local names for this site might be created so that attempts to send to oneself can be eliminated. These can either be defined directly in the configuration file or read in from another file. Classes may be given names from the set of upper case letters. Lower case letters and special characters are reserved for system use.

The syntax is:

C *c word1 word2*
F *c file*
F *c* **|** *command*

The first form defines the class *c* to match any of the named words. The third form executes the given command and reads the elements of the class from

standard output of the command. It is permissible to split them among multiple lines; for example, the two forms:

```
CHmonet ucbmonet
```

and

```
CHmonet
CHucbmonet
```

are equivalent.

M — Define Mailer

Programs and interfaces to mailers are defined in this line. The format is:

M*name* **,** {*field*=*value* }*

where *name* is the name of the mailer (used internally only) and the "field=name" pairs define attributes of the mailer. Fields are:

```
Path        The pathname of the mailer
Flags       Special flags for this mailer
Sender      A rewriting set for sender addresses
Recipient   A rewriting set for recipient addresses
Argv        An argument vector to pass to this mailer
Eol         The end-of-line string for this mailer
Maxsize     The maximum message length to this mailer
Length      The maximum length of the Argv for this mailer
```

Only the first character of the field name is checked.

H — Define Header

The format of the header lines that `sendmail` inserts into the message are defined by the H line. The syntax of this line is:

H [**?***mflags***?**] *hname* **:***htemplate*

Continuation lines in this spec are reflected directly into the outgoing message. The *htemplate* is macro expanded before insertion into the message. If the *mflags* (surrounded by question marks) are specified, at least one of the specified flags must be stated in the mailer definition for this header to be automatically output. If one of these headers is in the input it is reflected to the output regardless of these flags.

Some headers have special semantics that will be described below.

O — Set Option

There are a number of 'random' options that can be set from a configuration file. Options are represented by single characters. The syntax of this line is:

O*ovalue*

This sets option *o* to *value*. Depending on the option, *value* may be a string, an integer, a boolean (with legal values "t," "T," "f," or "F" — the default is TRUE), or a time interval. See D.7 for the list of options.

T — Define Trusted Users

Trusted users are those users who are permitted to override the sender address using the `-f` flag. These typically are "root," "uucp," and "network," but on some users it may be convenient to extend this list to include other users, perhaps to support a separate UUCP login for each host. The syntax of this line is:

T *user1 user2 . . .*

There may be more than one of these lines.

P — Precedence Definitions

Values for the "Precedence:" field may be defined using the **P** control line. The syntax of this field is:

P*name=num*

When the *name* is found in a "Precedence:" field, the message class is set to *num*. Higher numbers mean higher precedence. Numbers less than zero have the special property that error messages will not be returned. The default precedence is zero. For example, our list of precedences is:

```
Pfirst-class=0
Pspecial-delivery=100
Pjunk=-100
```

The Semantics

This section describes the details of rewriting rules and mailer descriptions.

Special Macros, Conditionals

Macros are interpolated using the construct **$***x*, where *x* is the name of the macro to be interpolated. In particular, lower case letters are reserved to have special semantics, used to pass information in or out of `sendmail`, and some special characters are reserved to provide conditionals, etc.

The following macros *must* be defined to transmit information into `sendmail`:

e	The SMTP entry message
j	The official domain name for this site
l	The format of the UNIX from line
n	The name of the daemon (for error messages)
o	The set of "operators" in addresses
q	default format of sender address

The **$e** macro is printed out when SMTP starts up. The first word must be the **$j** macro. The **$j** macro should be in RFC821 format. The **$l** and **$n** macros can be considered constants except under terribly unusual circumstances. The **$o** macro consists of a list of characters which will be considered tokens and which will separate tokens when doing parsing. For example, if "r" were in the **$o** macro, then the input "address" would be scanned as three tokens: "add," "r," and "ess." Finally, the **$q** macro specifies how an address should appear in a message when it is defaulted. For example, on our system these definitions are:

```
De$j Sendmail $v ready at $b
DnMAILER-DAEMON
DlFrom $g  $d
Do.:%@!^=/
Dq$g$?x ($x)$.
Dj$H.$D
```

An acceptable alternative for the **$q** macro is "$?x$x $.<$g>". These correspond to the following two formats:

```
eric@Berkeley (Eric Allman)
Eric Allman <eric@Berkeley>
```

Some macros are defined by `sendmail` for interpolation into argv's for mailers or for other contexts. These macros are:

a	The origination date in Arpanet format
b	The current date in Arpanet format
c	The hop count
d	The date in UNIX (ctime) format
f	The sender (from) address
g	The sender address relative to the recipient
h	The recipient host
i	The queue id
p	Sendmail's pid
r	Protocol used
s	Sender's host name
t	A numeric representation of the current time
u	The recipient user
v	The version number of sendmail
w	The hostname of this site
x	The full name of the sender
z	The home directory of the recipient

There are three types of dates that can be used. The **$a** and **$b** macros are in Arpanet format; **$a** is the time as extracted from the "Date:" line of the message (if there was one), and **$b** is the current date and time (used for postmarks). If no "Date:" line is found in the incoming message, **$a** is set to the current time also. The **$d** macro is equivalent to the **$a** macro in UNIX (ctime) format.

The **$f** macro is the id of the sender as originally determined; when mailing to a specific host the **$g** macro is set to the address of the sender *relative to the recipient*. For example, if I send to "bollard@matisse" from the machine "ucbarpa" the **$f** macro will be "eric" and the **$g** macro will be "eric@ucbarpa."

The **$x** macro is set to the full name of the sender. This can be determined in several ways. It can be passed as flag to `sendmail`. The second choice is the value of the "Full-name:" line in the header if it exists, and the third choice is the comment field of a "From:" line. If all of these fail, and if the message is being originated locally, the full name is looked up in the */etc/passwd* file.

When sending, the **$h**, **$u**, and $z macros get set to the host, user, and home directory (if local) of the recipient. The first two are set from the $@ and $: part of the rewriting rules, respectively.

The **$p** and **$t** macros are used to create unique strings (for example, for the "Message-Id:" field). The **$i** macro is set to the queue id on this host; if put into the timestamp line it can be extremely useful for tracking messages. The **$v** macro is set to be the version number of `sendmail`; this is normally put in timestamps and has been proven extremely useful for debugging. The **$w** macro is set to the name of this host if it can be determined. The **$c** field is set to the "hop count," that is, the number of times this message has been processed. This can be determined by the `-h` flag on the command line or by counting the timestamps in the message.

The **$r** and **$s** fields are set to the protocol used to communicate with `sendmail` and the sending hostname; these are not supported in the current version.

Conditionals can be specified using the syntax:

```
$?x text1 $| text2 $.
```

This interpolates *text1* if the macro **$x** is set, and *text2* otherwise. The "else" ($|) clause may be omitted.

Special Classes

The class **$=w** is set to be the set of all names this host is known by. This can be used to delete local hostnames.

The Left Hand Side

The left hand side of rewriting rules contains a pattern. Normal words are simply matched directly. Metasyntax is introduced using a dollar sign. The metasymbols are:

```
$*   Match zero or more tokens
$+   Match one or more tokens
$-   Match exactly one token
$=x  Match any token in class x
$~x  Match any token not in class x
$%x  Match any token in yp map $x
$!x  Match any token not in yp map $x
```

If any of these match, they are assigned to the symbol **$*n*** for replacement on the right hand side, where *n* is the index in the LHS. For example, if the LHS:

```
$-:$+
```

is applied to the input:

```
UCBARPA:eric
```

the rule will match, and the values passed to the RHS will be:

```
$1 UCBARPA
$2 eric
```

The **$%*x*** uses the macro *x* to specify the name of a `yp` map. The special form **$%y** matches any hostname in the 'hosts.byname' map, or in */etc/hosts* if not running `yp`.

The Right Hand Side

When the left hand side of a rewriting rule matches, the input is deleted and replaced by the right hand side. Tokens are copied directly from the RHS unless they begin with a dollar sign. Metasymbols are:

$*n*	Substitute indefinite token *n* from LHS
$>*n*	Call ruleset *n*
$#*mailer*	Resolve to *mailer*
$@*host*	Specify *host*
$:*user*	Specify *user*
$[*host*$]	Map to primary hostname

The **$*n*** syntax substitutes the corresponding value from a **$+**, **$–**, **$***, **$=**, or **$~** match on the LHS. It may be used anywhere.

The **$>*n*** syntax causes the remainder of the line to be substituted as usual and then passed as the argument to ruleset *n*. The final value of ruleset *n* then becomes the substitution for this rule.

The **$#** syntax should *only* be used in ruleset zero. It causes evaluation of the ruleset to terminate immediately, and signals to `sendmail` that the address has completely resolved. The complete syntax is:

$#*mailer*$@*host*$:*user*

This specifies the {mailer, host, user} 3-tuple necessary to direct the mailer. If the mailer is local the host part may be omitted. The *mailer* and *host* must be a single word, but the *user* may be multi-part.

A RHS may also be preceeded by a **$@** or a **$:** to control evaluation. A **$@** prefix causes the ruleset to return with the remainder of the RHS as the value. A **$:** prefix causes the rule to terminate immediately, but the ruleset to continue; this can be used to avoid continued application of a rule. The prefix is stripped before continuing.

The **$@** and **$:** prefixes may preceed a **$>** spec; for example:

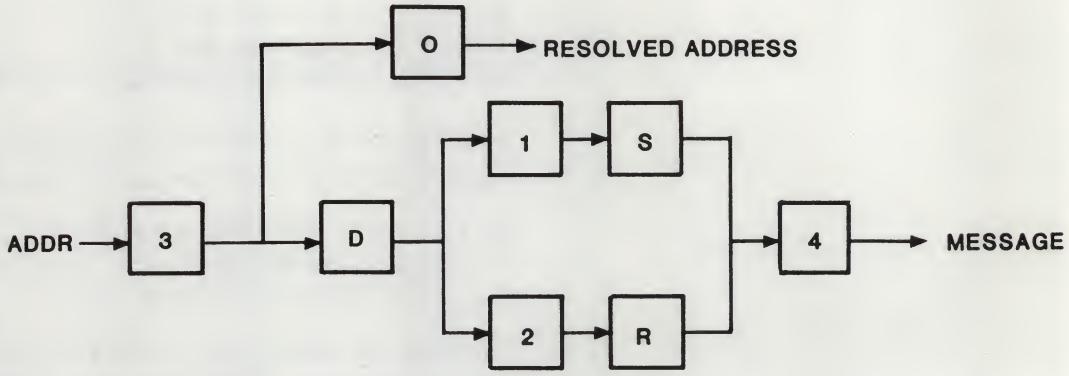
```
R$+     $:$>7$1
```

matches anything, passes that to ruleset seven, and continues; the **$:** is necessary to avoid an infinite loop.

Semantics of Rewriting Rule Sets

There are five rewriting sets that have specific semantics. These are related as depicted by Figure 1.

Figure D-1 *Rewriting Set Semantics*



D — sender domain addition
S — mailer-specific sender rewriting
R — mailer-specific recipient rewriting

Ruleset three should turn the address into "canonical form." This form should have the basic syntax:

```
local-part@host-domain-spec
```

If no "@" sign is specified, then the host-domain-spec *may* be appended from the sender address (if the **C** flag is set in the mailer definition corresponding to the *sending* mailer). Ruleset three is applied by `sendmail` before doing anything with any address.

Ruleset zero is applied after ruleset three to addresses that are going to actually specify recipients. It must resolve to a *{mailer, host, user}* triple. The *mailer* must be defined in the mailer definitions from the configuration file. The *host* is defined into the **$h** macro for use in the argv expansion of the specified mailer; the *user* is defined into **$u.**

Rulesets one and two are applied to all **from:** and **to:** and **cc:** recipient addresses respectively. Then the rulesets specified in the mailer definition line (S= and R=) are applied. Note that this will be done many times for one message, depending on how many mailers the message is routed to by ruleset zero.

Ruleset four is applied last to all addresses in the message. It is typically used to translate internal to external form.

Mailer Flags, etc.

There are a number of flags that may be associated with each mailer, each identified by a letter of the alphabet. Many of them are assigned semantics internally. These are detailed in D.8. Any other flags may be used freely to conditionally assign headers to messages destined for particular mailers.

The ''error'' Mailer

The mailer with the special name ''error'' can be used to generate a user error. The (optional) host field is a numeric exit status to be returned, and the user field is a message to be printed. For example, the entry:

```
$#error$:Host unknown in this domain
```

on the RHS of a rule will cause the specified error to be generated if the LHS matches. This mailer is only functional in ruleset zero.

Semantics of Mailer Descriptions

Each mailer has an internal name. This can be arbitrary, except that the names ''local'' and ''prog'' must be defined. Ruleset zero will resolve addresses to this mailer name (and a host and user name).

The pathname of the mailer must be given in the P field. If this mailer should be accessed via a TCP connection, use the string ''[IPC]'' instead.

The F field defines the mailer flags. You should specify an ''f'' or ''r'' flag to pass the name of the sender as a `-f` or `-r` flag respectively. These flags are only passed if they were passed to `sendmail`, so that mailers that give errors under some circumstances can be placated. If the mailer is not picky you can just specify ''–f $g'' in the argv template. If the mailer must be called as **root** the ''S'' flag should be given; this will not reset the userid before calling the mailer.[9] If this mailer is local (that is, will perform final delivery rather than another network hop) the ''l'' flag should be given. Quote characters (backslashes and " marks) can be stripped from addresses if the ''s'' flag is specified; if this is not given they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction the ''m'' flag should be stated. If this flag is on, then the argv template containing **$u** will be repeated for each unique user on a given host. The ''e'' flag will mark the mailer as being 'expensive,' which will cause `sendmail` to defer connection until a queue run.[10]

An unusual case is the ''C'' flag. This flag applies to the mailer that the message is received from, rather than the mailer being sent to; if set, the domain spec of the sender (that is, the ''@host.domain'' part) is saved and is appended to any addresses in the message that do not already contain a domain spec. For example, a message of the form:

```
From: eric@ucbarpa
To: wnj@monet, mckusick
```

will be modified to:

---

[9] `sendmail` must be running setuid to root for this to work.

[10] The **c** configuration option must be given for this to be effective.

```
From: eric@ucbarpa
To: wnj@monet, mckusick@ucbarpa
```

if and only if the "C" flag is defined in the mailer corresponding to "eric@ucbarpa."

Other flags are described in D.8.

The S and R fields in the mailer description are per-mailer rewriting sets to be applied to sender and recipient addresses respectively. These are applied after the sending domain is appended and the general rewriting sets (number one or two) are applied, but before the output rewrite (ruleset four) is applied. A typical use is to append the current domain to addresses that do not already have a domain. For example, a header of the form:

```
From: eric
```

might be changed to be:

```
From: eric@ucbarpa
```

or

```
From: ucbvax!eric
```

depending on the domain it is being shipped into. These sets can also be used to do special purpose output rewriting in cooperation with ruleset four.

The E field defines the string to use as an end-of-line indication. A string containing only newline is the default. The usual backslash escapes (\r, \n, \f, \b) may be used.

An argv template is given as the A field. It may have embedded spaces. The template is macro-expanded before being passed to the mailer. Useful macros include **$h,** the host name resolved by ruleset zero, and **$u,** the user name (or names) resolved. If there is no argv with a **$u** macro in it, `sendmail` will speak SMTP to the mailer. If the pathname for this mailer is "[IPC]," the argv should be

```
IPC $h [ port ]
```

where *port* is the optional port number to connect to.

If an L field exists, it specifies the maximum length of the **$u** macro in the argv passed to the mailer. This can be used with the **m** flag to send multiple recipients with one call to the mailer, while avoiding mailer limitations on argument length. This makes *UUCP* mail more efficient. **$u** will always expand to at least one recipient even if that recipient exceeds the L= limit.

For example, the specification:

```
Mlocal, P=/bin/mail, F=rlsm  S=10, R=20, A=mail -d $u
Mether, P=[IPC],     F=meC,  S=11, R=21, A=IPC $h, M=100000
```

specifies a mailer to do local delivery and a mailer for ethernet delivery. The first is called "local," is located in the file */bin/mail*, takes a picky `-r` flag, does local delivery, quotes should be stripped from addresses, and multiple users can be delivered at once; ruleset ten should be applied to sender addresses in the

message and ruleset twenty should be applied to recipient addresses; the argv to send to a message will be the word "mail," the word "–d," and words containing the name of the receiving user. If a `-r` flag is inserted it will be between the words "mail" and "–d." The second mailer is called "ether," it should be connected to via an IPC connection, it can handle multiple users at once, connections should be deferred, and any domain from the sender address should be appended to any receiver name without a domain; sender addresses should be processed by ruleset eleven and recipient addresses by ruleset twenty-one. There is a 100,000 byte limit on messages passed through this mailer.

Building a Configuration File from Scratch

Building a configuration table from scratch is an extremely difficult job. Fortunately, it is almost never necessary to do so; nearly every situation that may come up may be resolved by changing an existing table. In any case, it is critical that you understand what it is that you are trying to do and come up with a philosophy for the configuration table. This section is intended to explain what the real purpose of a configuration table is and to give you some ideas for what your philosophy might be.

What you are Trying to do

The configuration table has three major purposes. The first and simplest is to set up the environment for `sendmail`. This involves setting the options, defining a few critical macros, etc. Since these are described in other places, we will not go into more detail here.

The second purpose is to rewrite addresses in the message. This should typically be done in two phases. The first phase maps addresses in any format into a canonical form. This should be done in ruleset three. The second phase maps this canonical form into the syntax appropriate for the receiving mailer. `sendmail` does this in three subphases. Rulesets one and two are applied to all sender and recipient addresses respectively. After this, you may specify per-mailer rulesets for both sender and recipient addresses; this allows mailer-specific customization. Finally, ruleset four is applied to do any default conversion to external form.

The third purpose is to map addresses into the actual set of instructions necessary to get the message delivered. Ruleset zero must resolve to the internal form, which is in turn used as a pointer to a mailer descriptor. The mailer descriptor describes the interface requirements of the mailer.

Philosophy

The particular philosophy you choose will depend heavily on the size and structure of your organization. I will present a few possible philosophies here.

One general point applies to all of these philosophies: it is almost always a mistake to try to do full name resolution. For example, if you are trying to get names of the form "user@host" to the Arpanet, it does not pay to route them to "sun!snail!athena!c70:user@host" since you then depend on several links not under your control. The best approach to this problem is to simply forward to "sun!user@host" and let sun worry about it from there. In summary, just get the message closer to the destination, rather than determining the full path.

Large Site, Many Hosts — Minimum Information

Sun is an example of a large site; it has more than two or three hosts. Sun has decided that the only reasonable philosophy for their environment is to designate one host as site guru. It must be able to resolve any piece of mail it receives. The other sites should have the minimum amount of information they can get away with, and even this minimum should be hints rather than solid information.

For example, a typical site on the Sun local ether network is "athena." Athena has a list of known ethernet hosts; if it receives mail for any of them, it can do direct delivery. If it receives mail for any unknown host, it just passes it directly to "snail," the Sun master host. Snail may determine that the host name is illegal and reject the message, or may be able to do delivery. However, it is important to note that when a new ethernet host is added, the only host that *must* have its tables updated is snail; the others *may* be updated as convenient, but this is not critical.

This picture is slightly muddied due to network connections that are not actually located on snail. For example, the Sun TCP connection is currently on "moonshine." However, athena *does not* know about this; the information is hidden between snail and moonshine. Mail going from athena to a TCP host is transferred via the ethernet from athena to snail, then via the ethernet from snail to moonshine, and then is submitted to the Arpanet. Although this involves some extra hops, Sun feels this is an acceptable tradeoff.

An interesting point is that it would be possible to update athena to send TCP mail directly to moonshine if the load got too high: if athena failed to note a host as a TCP host, the mail would go via snail as before; and if athena incorrectly sent a message to moonshine, it would still be sent by moonshine to snail as before. The only problem that might occur in this scenario is 'looping': if moonshine thought that snail had the TCP connection and vice versa. For this reason, updates should *always* happen to the master host first.

This philosophy results as much from the need to have a single source for the configuration files (typically built using *m4* (1) or some similar tool) as any logical need. Maintaining more than three separate tables by hand is essentially an impossible job.

Small Site — Complete Information

A small site (two or three hosts) may find it more reasonable to have complete information at each host. This would require that each host know exactly where each network connection is, possibly including the names of each host on that network. As long as the site remains small and the the configuration remains relatively static, the update problem will probably not be too great.

Single Host

This is in some sense the trivial case. The only major issue is trying to insure that you don't have to know too much about your environment. For example, if you have a UUCP connection you might find it useful to know about the names of hosts connected directly to you, but this is really not necessary since this may be determined from the syntax.

Relevant Issues

The canonical form you use should almost certainly be as specified in the Arpanet protocols RFC819 and RFC822.

RFC822 describes the format of the mail message itself. `sendmail` follows this RFC closely, to the extent that many of the standards described in this document can not be changed without changing the code. In particular, the following characters have special interpretations:

```
< > ( ) " \
```

Any attempt to use these characters for other than their RFC822 purpose in addresses is probably doomed to disaster.

RFC819 describes the specifics of the domain-based addressing. This is touched on in RFC822 as well. Essentially each host is given a name which is a right-to-left dot qualified pseudo-path from a distinguished root. The elements of the path need not be physical hosts; the domain is logical rather than physical. For example, at Sun one legal host is ''a.b1.silicon.arpa''. Reading from right to left, ''arpa'' is a top level domain (related to, but not limited to, the physical Arpanet), ''silicon'' is both an Arpanet host and a logical domain which is actually interpreted by a host called snail (the ''major'' host for this domain), ''b1'' represents the Building One, (in this case a strictly logical entity), and ''a'' is a host in Building One; this particular host happens to be connected via ethernet, but other hosts might be connected via some other network.

Be aware when reading RFC819 that there are a number of errors in it.

How to Proceed

Once you have decided on a philosophy, it is worth examining the available configuration tables to decide if any of them are close enough to steal major parts of. Even under the worst of conditions, there is a fair amount of boiler plate that can be collected safely.

The next step is to build ruleset three. This will be the hardest part of the job. Beware of doing too much to the address in this ruleset, since anything you do will reflect through to the message. In particular, stripping of local domains is best deferred, since this can leave you with addresses with no domain spec at all. Since `sendmail` likes to append the sending domain to addresses with no domain, this can change the semantics of addresses. Also try to avoid fully qualifying domains in this ruleset. Although technically legal, this can lead to unpleasantly and unnecessarily long addresses reflected into messages. The Sun configuration files define ruleset nine to qualify domain names and strip local domains. This is called from ruleset zero to get all addresses into a cleaner form.

Once you have ruleset three finished, the other rulesets should be relatively trivial. If you need hints, examine the supplied configuration tables.

Testing the Rewriting Rules — the –bt Flag

When you build a configuration table, you can do a certain amount of testing using the ''test mode'' of `sendmail`. For example, you could invoke `sendmail` as:

```
sendmail -bt -Ctest.cf
```

which would read the configuration file *test.cf* and enter test mode. In this mode,

you enter lines of the form:

```
rwset address
```

where *rwset* is the rewriting set you want to use and *address* is an address to apply the set to. Test mode shows you the steps it takes as it proceeds, finally showing you the address it ends up with. You may use a comma separated list of *rwsets* for sequential application of rules to an input; ruleset three is always applied first. For example:

```
1,21,4 monet:bollard
```

first applies ruleset three to the input "monet:bollard." Ruleset one is then applied to the output of ruleset three, followed similarly by rulesets twenty-one and four.

If you need more detail, you can also use the `-d21` flag to turn on more debugging. For example,

```
sendmail -bt -d21.99
```

turns on an incredible amount of information; a single word address is probably going to print out several pages worth of information.

D.6. Command Line Flags

Arguments must be presented with flags before addresses. The flags are:

–f *addr* The sender's machine address is *addr*. This flag is ignored unless the real user is listed as a "trusted user" or if *addr* contains an exclamation point (because of certain restrictions in UUCP).

–r *addr* An obsolete form of `-f`.

–h *cnt* Sets the "hop count" to *cnt*. This represents the number of times this message has been processed by `sendmail` (to the extent that it is supported by the underlying networks). *Cnt* is incremented during processing, and if it reaches MAXHOP (currently 30) `sendmail` throws away the message with an error.

–F*name* Sets the full name of this user to *name*.

–n Don't do aliasing or forwarding.

–t Read the header for "To:," "Cc:," and "Bcc:" lines, and send to everyone listed in those lists. The "Bcc:" line will be deleted before sending. Any addresses in the argument vector will be deleted from the send list.

–b*x* Set operation mode to *x*. Operation modes are:

```
m   Deliver mail (default)
a   Run in arpanet mode (see below)
s   Speak SMTP on input side
d   Run as a daemon
t   Run in test mode
v   Just verify addresses, don't collect or deliv
i   Initialize the alias database
p   Print the mail queue
z   Freeze the configuration file
```

The special processing for the ARPANET includes reading the "From:" line from the header to find the sender, printing ARPANET style messages (preceded by three digit reply codes for compatibility with the FTP protocol [Neigus73, Postel74, Postel77]), and ending lines of error messages with <CRLF>.

–q*time* Try to process the queued up mail. If the time is given, a `sendmail` will run through the queue at the specified interval to deliver queued mail; otherwise, it only runs once.

–C*file* Use a different configuration file.

–d*level* Set debugging level.

–o*xvalue* Set option *x* to the specified *value*. These options are described in D.7.

There are a number of options that may be specified as primitive flags (provided for compatibility with *delivermail*). These are the e, i, m, and v options. Also,

the f option may be specified as the `-s` flag.

D.7. Configuration Options

The following options may be set using the `-o` flag on the command line or the **O** line in the configuration file:

A*file* — Use the named *file* as the alias file instead of */usr/lib/aliases*. If no file is specified, use *aliases* in the current directory.

a*time* — If set, time to wait for an "@:@" entry to exist in the alias database before starting up. If it does not appear after that time, rebuild the database.

B*value* — Blank substitute. Default is "."

C*n* — Checkpoint after *n* recipients.

c — If an outgoing mailer is marked as being expensive, don't connect immediately. This requires that queueing be compiled in, since it will depend on a queue run process to actually send the mail.

d*x* — Deliver in mode *x*. Legal modes are:

```
i   Deliver interactively (synchronously)
b   Deliver in background (asynchronously)
q   Just queue the message (deliver during queue run)
```

D — If set, rebuild the alias database if necessary and possible. If this option is not set, `sendmail` will never rebuild the alias database unless explicitly requested using `-bi`.

e*x* — Dispose of errors using mode *x*. The values for *x* are:

```
p   Print error messages (default)
q   No messages, just give exit status
m   Mail back errors
w   Write back errors (mail if user not logged in)
e   Mail back errors and give zero exit stat always
```

F*n* — The temporary queue file mode, in octal. 644 and 600 are good choices.

f — Save UNIX-style "From" lines at the front of headers. Normally they are assumed redundant and discarded.

g*n* — Set the default group id for mailers to run in to *n*.

H*file* — Specify the help file for SMTP.

i — Ignore dots in incoming messages.

L*n* — Set the default log level to *n*.

M*xvalue* — Set the macro *x* to *value*. This is intended only for use from the command line.

m — Send to me too, even if I am in an alias expansion.

o — Assume that the headers may be in old format, that is, spaces delimit names. This actually turns on an adaptive algorithm: if any recipient address contains a comma, parenthesis, or angle bracket, it will be

	assumed that commas already exist. If this flag is not on, only commas delimit names. Headers are always output with commas between the names.
P	The name of the local Postmaster. If defined, error messages from the MAILER-DAEMON cause the header to be sent to this address.
Q*dir*	Use the named *dir* as the queue directory.
q*limit*	Size limit of messages to be queued under heavy load. Default is 10,000 bytes.
r*time*	Timeout reads after *time* interval.
S*file*	Log statistics in the named *file*.
s	Be super-safe when running things, that is, always instantiate the queue file, even if you are going to attempt immediate delivery. `sendmail` always instantiates the queue file before returning control the the client under any circumstances.
T*time*	Set the queue timeout to *time*. After this interval, messages that have not been successfully sent will be returned to the sender.
u*n*	Set the default userid for mailers to *n*. Mailers without the *S* flag in the mailer definition will run as this user.
v	Run in verbose mode.
x	Set the load average value which causes `sendmail` to simply queue mail (regardless of the `d`*x* option) to reduce system load. Default is 8.
X	Set the load average value which causes the `sendmail` daemon to refuse incoming SMTP connections to reduce system load. Default is 12.
Y	Yellow pages map name to be used for aliases. Default is *mail.aliases*.

D.8. Mailer Flags

The following flags may be set in the mailer description.

C If mail is *received* from a mailer with this flag set, any addresses in the header that do not have an at sign (''@'') after being rewritten by ruleset three will have the ''@domain'' clause from the sender tacked on. This allows mail with headers of the form:

```
From: usera@hosta
To: userb@hostb, userc
```

to be rewritten as:

```
From: usera@hosta
To: userb@hostb, userc@hosta
```

automatically.

D This mailer wants a ''Date:'' header line.

E Escape ''From'' lines to be ''>From'' (usually specified with `U`.

e This mailer is expensive to connect to, so try to avoid connecting normally; any necessary connection will occur during a queue run.

F This mailer wants a ''From:'' header line.

f The mailer wants a `-f` *from* flag, but only if this is a network forward operation (that is, the mailer will give an error if the executing user does not have special permissions).

h Upper case should be preserved in host names for this mailer.

L Limit the line lengths as specified in RFC821.

l This mailer is local (that is, final delivery will be performed).

M This mailer wants a ''Message-Id:'' header line.

m This mailer can send to multiple users on the same host in one transaction. When a **$u** macro occurs in the *argv* part of the mailer definition, that field will be repeated as necessary for all qualifying users. The L= field of the mailer description can be used to limit the total length of the **$u** expansion.

n Do not insert a UNIX-style ''From'' line on the front of the message.

P This mailer wants a ''Return-Path:'' line.

p Always add local host name to the ''MAIL From:'' line of *SMTP*, even if there already is one.

r Same as f, but sends a `-r` flag.

S Don't reset the userid before calling the mailer. This would be used in a secure environment where `sendmail` ran as root. This could be used to avoid forged addresses.

s Strip quote characters off of the address before calling the mailer.

U This mailer wants UNIX-style ''From'' lines with the ugly UUCP-style ''remote from <host>'' on the end.

u Upper case should be preserved in user names for this mailer.

X This mailer want to use the hidden dot algorithm as specified in RFC821; basically, any line beginning with a dot will have an extra dot prepended (to be stripped at the other end). This insures that lines in the message containing a dot will not terminate the message prematurely.

x This mailer wants a "Full-Name:" header line.

D.9. Summary of Support Files

This is a summary of the support files that `sendmail` creates, uses, or generates.

/usr/lib/sendmail	The binary of `sendmail`.
/usr/ucb/newaliases	A link to /usr/lib/sendmail; causes the alias database to be rebuilt. Running this program is completely equivalent to giving `sendmail` the `-bi` flag.
/usr/lib/sendmail.cf	The configuration file, in textual form.
/usr/lib/sendmail.fc	The frozen configuration file.
/usr/lib/sendmail.hf	The SMTP help file.
/usr/lib/aliases	The textual version of the alias file.
/usr/lib/aliases.{pag,dir}	The alias file in *dbm*(3) format.
/usr/etc/in.syslog	The program to do logging.
/etc/syslog.conf	The configuration file for syslog.
/etc/syslog.pid	Contains the process id of the currently running syslog.
/usr/spool/mqueue	The directory in which the mail queue and temporary files reside.
/usr/spool/mqueue/qf*	Control (queue) files for messages.
/usr/spool/mqueue/df*	Data files.
/usr/spool/mqueue/lf*	Lock files
/usr/spool/mqueue/tf*	Temporary versions of the qf files, used during queue file rebuild.
/usr/spool/mqueue/nf*	A file used when creating a unique id.
/usr/spool/mqueue/xf*	A transcript of the current session.

E

sendmail — Internetwork Mail Router

sendmail — Internetwork Mail Router **365**

E.1. Design Goals 366
E.2. Overview 368
System Organization 368
Interfaces to the Outside World 368
Operational Description 368
Message Header Editing 369
Configuration File 370
E.3. Usage and Implementation 370
Arguments 370
Mail to Files and Programs 370
Aliasing, Forwarding, Inclusion 371
Message Collection 371
Message Delivery 372
Queued Messages 372
Configuration 372
E.4. Evaluations and Future Plans 373

E

sendmail — Internetwork Mail Router

Routing mail through a heterogenous internet presents many new problems. Among the worst of these is that of address mapping. Historically, this has been handled on an *ad hoc* basis. However, this approach has become unmanageable as internets grow.

`sendmail` acts a unified "post office" to which all mail can be submitted. Address interpretation is controlled by a production system, which can parse both domain-based addressing and old-style *ad hoc* addresses. The production system is powerful enough to rewrite addresses in the message header to conform to the standards of a number of common target networks, (TCP/RFC822) Arpa Internet, UUCP, and Phonenet. Sendmail also implements an SMTP server, message queueing, and aliasing.

`sendmail` implements a general internetwork mail routing facility, featuring aliasing and forwarding, automatic routing to network gateways, and flexible configuration.

In a simple network, each node has an address, and resources can be identified with a host-resource pair; in particular, the mail system can refer to users using a host-username pair. Host names and numbers have to be administered by a central authority, but usernames can be assigned locally to each host.

In an internet, multiple networks with different characterstics and managements must communicate. In particular, the syntax and semantics of resource identification change. Certain special cases can be handled trivially by *ad hoc* techniques, such as providing network names that appear local to hosts on other networks, as with the Ethernet at Xerox PARC. However, the general case is extremely complex. For example, some networks require point-to-point routing, which simplifies the database update problem since only adjacent hosts must be entered into the system tables, while others use end-to-end addressing. Some networks use a left-associative syntax and others use a right-associative syntax, causing ambiguity in mixed addresses.

Internet standards seek to eliminate these problems. Initially, the standards proposed expanding the address pairs to address triples, consisting of {network, host, resource} triples. Network numbers must be universally agreed upon, and hosts can be assigned locally on each network. The user-level presentation was quickly expanded to address domains, comprised of a local resource identification and a hierarchical domain specification with a common static root.

The domain technique separates the issue of physical versus logical addressing. For example, an address of the form "eric@a.cc.berkeley.arpa" describes only the logical organization of the address space.

`sendmail` is intended to help bridge the gap between the totally *ad hoc* world of networks that know nothing of each other and the clean, tightly-coupled world of unique network numbers. It can accept old arbitrary address syntaxes, resolving ambiguities using heuristics specified by the system administrator, as well as domain-based addressing. It helps guide the conversion of message formats between disparate networks. In short, `sendmail` is designed to assist a graceful transition to consistent internetwork addressing schemes.

Section E.1 discusses the design goals for `sendmail`. Section E.2 gives an overview of the basic functions of the system. Section E.3 discusses details of usage. Section E.4 gives an evaluation of `sendmail`, including future plans.

E.1. Design Goals

Design goals for `sendmail` include:

1. Compatibility with the existing mail programs, including Bell version 6 mail, Bell version 7 mail [UNIX83], Berkeley *Mail* [Shoens79], BerkNet mail [Schmidt79], and UUCP mail [Nowitz78a, Nowitz78b]. ARPANET mail [Crocker77a, Postel77] was also required.

2. Reliability, in the sense of guaranteeing that every message is correctly delivered or at least brought to the attention of a human for correct disposal; no message should ever be completely lost. This goal was considered essential because of the emphasis on mail in our environment. It has turned out to be one of the hardest goals to satisfy, especially in the face of the many anomalous message formats produced by various ARPANET sites. For example, certain sites generate improperly formated addresses, occasionally causing error-message loops. Some hosts use blanks in names, causing problems with UNIX mail programs that assume that an address is one word. The semantics of some fields are interpreted slightly differently by different sites. In summary, the obscure features of the ARPANET mail protocol really *are* used and are difficult to support, but must be supported.

3. Existing software to do actual delivery should be used whenever possible. This goal derives as much from political and practical considerations as technical.

4. Easy expansion to fairly complex environments, including multiple connections to a single network type (such as with multiple UUCP or Ether nets [Metcalfe76]). This goal requires consideration of the contents of an address as well as its syntax in order to determine which gateway to use. For example, the ARPANET is bringing up the TCP protocol to replace the old NCP protocol. No host at Berkeley runs both TCP and NCP, so it is necessary to look at the ARPANET host name to determine whether to route mail to an NCP gateway or a TCP gateway.

5. Configuration should not be compiled into the code. A single compiled program should be able to run as is at any site (barring such basic changes as

the CPU type or the operating system). We have found this seemingly unimportant goal to be critical in real life. Besides the simple problems that occur when any program gets recompiled in a different environment, many sites like to "fiddle" with anything that they will be recompiling anyway.

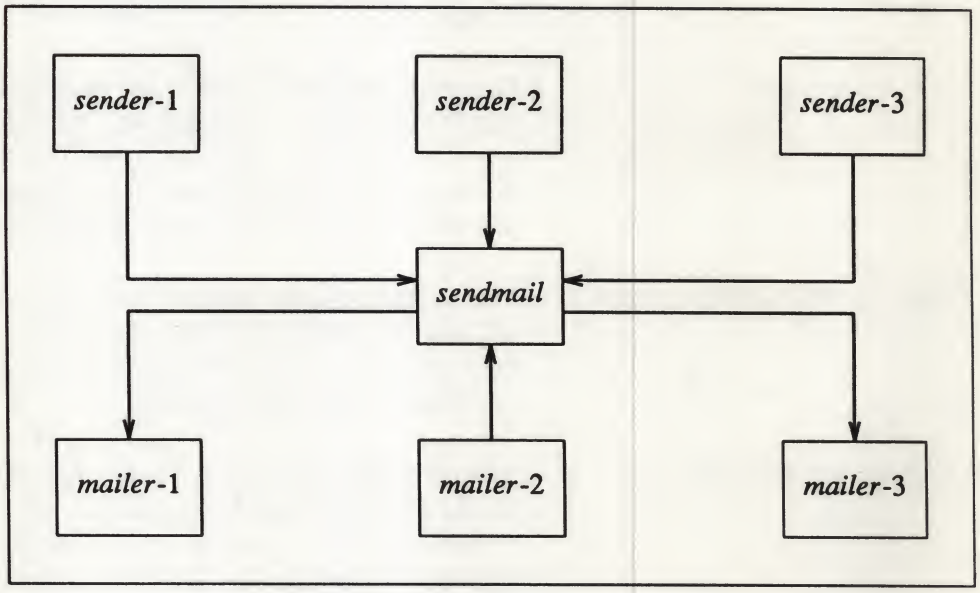
6. `sendmail` must be able to let various groups maintain their own mailing lists, and let individuals specify their own forwarding, without modifying the system alias file.

7. Each user should be able to specify which mailer to execute to process mail being delivered for him. This feature allows users who are using specialized mailers that use a different format to build their environment without changing the system, and facilitates specialized functions (such as returning an "I am on vacation" message).

8. Network traffic should be minimized by batching addresses to a single host where possible, without assistance from the user.

These goals motivated the architecture illustrated in Figure 1.

Figure E-1 *Sendmail System Structure*



The user interacts with a mail generating and sending program. When the mail is created, the generator calls `sendmail`, which routes the message to the correct mailer(s). Since some of the senders may be network servers and some of the mailers may be network clients, `sendmail` may be used as an internet mail gateway.

E.2. Overview

System Organization

`sendmail` neither interfaces with the user nor does actual mail delivery. Rather, it collects a message generated by a user interface program (UIP) such as Berkeley *Mail*, MS [Crocker77b], or MH [Borden79], edits the message as required by the destination network, and calls appropriate mailers to do mail delivery or queueing for network transmission[11]. This discipline allows the insertion of new mailers at minimum cost. In this sense `sendmail` resembles the Message Processing Module (MPM) of [Postel79b].

Interfaces to the Outside World

There are three ways `sendmail` can communicate with the outside world, both in receiving and in sending mail. These are using the conventional UNIX argument vector/return status, speaking SMTP over a pair of UNIX pipes, and speaking SMTP over an interprocess(or) channel.

Argument vector/exit status

This technique is the standard UNIX method for communicating with the process. A list of recipients is sent in the argument vector, and the message body is sent on the standard input. Anything that the mailer prints is simply collected and sent back to the sender if there were any problems. The exit status from the mailer is collected after the message is sent, and a diagnostic is printed if appropriate.

SMTP over Pipes

The SMTP protocol [Postel82] can be used to run an interactive lock-step interface with the mailer. A subprocess is still created, but no recipient addresses are passed to the mailer via the argument list. Instead, they are passed one at a time in commands sent to the processes standard input. Anything appearing on the standard output must be a reply code in a special format.

SMTP over an IPC Connection

This technique is similar to the previous technique, except that it uses a 4.2BSD IPC channel [UNIX83]. This method is exceptionally flexible in that the mailer need not reside on the same machine. It is normally used to connect to a sendmail process on another machine.

Operational Description

When a sender wants to send a message, it issues a request to `sendmail` using one of the three methods described above. `sendmail` operates in two distinct phases. In the first phase, it collects and stores the message. In the second phase, message delivery occurs. If there were errors during processing during the second phase, `sendmail` creates and returns a new message describing the error and/or returns an status code telling what went wrong.

Argument Processing and Address Parsing

If `sendmail` is called using one of the two subprocess techniques, the arguments are first scanned and option specifications are processed. Recipient addresses are then collected, either from the command line or from the SMTP RCPT command, and a list of recipients is created. Aliases are expanded at this step, including mailing lists. As much validation as possible of the addresses is

[11] except when mailing to a file, when `sendmail` does the delivery directly.

done at this step: syntax is checked, and local addresses are verified, but detailed checking of host names and addresses is deferred until delivery. Forwarding is also performed as the local addresses are verified.

`sendmail` appends each address to the recipient list after parsing. When a name is aliased or forwarded, the old name is retained in the list, and a flag is set that tells the delivery phase to ignore this recipient. This list is kept free from duplicates, preventing alias loops and duplicate messages deliverd to the same recipient, as might occur if a person is in two groups.

Message Collection

`sendmail` then collects the message. The message should have a header at the beginning. No formatting requirements are imposed on the message except that they must be lines of text (in other words, binary data is not allowed). The header is parsed and stored in memory, and the body of the message is saved in a temporary file.

To simplify the program interface, the message is collected even if no addresses were valid. The message will be returned with an error.

Message Delivery

For each unique mailer and host in the recipient list, `sendmail` calls the appropriate mailer. Each mailer invocation sends to all users receiving the message on one host. Mailers that only accept one recipient at a time are handled properly.

The message is sent to the mailer using one of the same three interfaces used to submit a message to sendmail. Each copy of the message is prepended by a customized header. The mailer status code is caught and checked, and a suitable error message given as appropriate. The exit code must conform to a system standard or a generic message (''Service unavailable'') is given.

Queueing for Retransmission

If the mailer returned an status that indicated that it might be able to handle the mail later, `sendmail` will queue the mail and try again later.

Return to Sender

If errors occur during processing, `sendmail` returns the message to the sender for retransmission. The letter can be mailed back or written in the *dead.letter* file in the sender's home directory[12].

Message Header Editing

Certain editing of the message header occurs automatically. Header lines can be inserted under control of the configuration file. Some lines can be merged; for example, a ''From:'' line and a ''Full-name:'' line can be merged under certain circumstances.

[12] Obviously, if the site giving the error is not the originating site, the only reasonable option is to mail back to the sender. Also, there are many more error disposition options, but they only effect the error message — the ''return to sender'' function is always handled in one of these two ways.

Configuration File

Almost all configuration information is read at runtime from an ASCII file, encoding macro definitions (defining the value of macros used internally), header declarations (telling sendmail the format of header lines that it will process specially, for example, lines that it will add or reformat), mailer definitions (giving information such as the location and characteristics of each mailer), and address rewriting rules (a limited production system to rewrite addresses which is used to parse and rewrite the addresses).

To improve performance when reading the configuration file, a memory image can be provided. This provides a ''compiled'' form of the configuration file.

E.3. Usage and Implementation

Arguments

Arguments may be flags and addresses. Flags set various processing options. Following flag arguments, address arguments may be given, unless we are running in SMTP mode. Addresses follow the syntax in RFC822 [Crocker82] for ARPANET address formats. In brief, the format is:

1. Anything in parentheses is thrown away (as a comment).
2. Anything in angle brackets (''< >'') is preferred over anything else. This rule implements the ARPANET standard that addresses of the form

   user name <machine-address>

   will send to the electronic ''machine-address'' rather than the human ''user name.''
3. Double quotes (") quote phrases; backslashes quote characters. Backslashes are more powerful in that they will cause otherwise equivalent phrases to compare differently — for example, *user* and "*user*" are equivalent, but *\user* is different from either of them.

Parentheses, angle brackets, and double quotes must be properly balanced and nested. The rewriting rules control remaining parsing[13].

Mail to Files and Programs

Files and programs are legitimate message recipients. Files provide archival storage of messages, useful for project administration and history. Programs are useful as recipients in a variety of situations, for example, to maintain a public repository of systems messages (such as the Berkeley *msgs* program, or the MARS system [Sattley78]).

Any address passing through the initial parsing algorithm as a local address (i.e, not appearing to be a valid address for another mailer) is scanned for two special cases. If prefixed by a vertical bar (''|'') the rest of the address is processed as a shell command. If the user name begins with a slash mark (''/'') the name is used as a file name, instead of a login name.

Files that have setuid or setgid bits set but no execute bits set have those bits honored if `sendmail` is running as root.

---

[13] Disclaimer: Some special processing is done after rewriting local names; see below.

Aliasing, Forwarding, Inclusion

`sendmail` reroutes mail three ways. Aliasing applies system wide. Forwarding allows each user to reroute incoming mail destined for that account. Inclusion directs `sendmail` to read a file for a list of addresses, and is normally used in conjunction with aliasing.

Aliasing

Aliasing maps names to address lists using a system-wide file. This file is indexed to speed access. Only names that parse as local are allowed as aliases; this guarantees a unique key (since there are no nicknames for the local host).

Forwarding

After aliasing, recipients that are local and valid are checked for the existence of a ".forward" file in their home directory. If it exists, the message is *not* sent to that user, but rather to the list of users in that file. Often this list will contain only one address, and the feature will be used for network mail forwarding.

Forwarding also permits a user to specify a private incoming mailer. For example, forwarding to:

```
"|/usr/local/newmail myname"
```

will use a different incoming mailer.

Inclusion

Inclusion is specified in RFC 733 [Crocker77a] syntax:

```
:Include: pathname
```

An address of this form reads the file specified by *pathname* and sends to all users listed in that file.

The intent is *not* to support direct use of this feature, but rather to use this as a subset of aliasing. For example, an alias of the form:

```
project: :include:/usr/project/userlist
```

is a method of letting a project maintain a mailing list without interaction with the system administration, even if the alias file is protected.

It is not necessary to rebuild the index on the alias database when a :include: list is changed.

Message Collection

Once all recipient addresses are parsed and verified, the message is collected. The message comes in two parts: a message header and a message body, separated by a blank line.

The header is formatted as a series of lines of the form

```
field-name: field-value
```

Field-value can be split across lines by starting the following lines with a space or a tab. Some header fields have special internal meaning, and have appropriate special processing. Other headers are simply passed through. Some header fields may be added automatically, such as time stamps.

The body is a series of text lines. It is completely uninterpreted and untouched, except that lines beginning with a dot have the dot doubled when transmitted over an SMTP channel. This extra dot is stripped by the receiver.

Message Delivery

The send queue is ordered by receiving host before transmission to implement message batching. Each address is marked as it is sent so rescanning the list is safe. An argument list is built as the scan proceeds. Mail to files is detected during the scan of the send list. The interface to the mailer is performed using one of the techniques described in section 2.2.

After a connection is established, `sendmail` makes the per-mailer changes to the header and sends the result to the mailer. If any mail is rejected by the mailer, a flag is set to invoke the return-to-sender function after all delivery completes.

Queued Messages

If the mailer returns a "temporary failure" exit status, the message is queued. A control file is used to describe the recipients to be sent to and various other parameters. This control file is formatted as a series of lines, each describing a sender, a recipient, the time of submission, or some other salient parameter of the message. The header of the message is stored in the control file, so that the associated data file in the queue is just the temporary file that was originally collected.

Configuration

Configuration is controlled primarily by a configuration file read at startup. `sendmail` should not need to be recomplied except

1. To change operating systems (V6, V7/32V, 4BSD).
2. To remove or insert the DBM (UNIX database) library.
3. To change ARPANET reply codes.
4. To add headers fields requiring special processing.

Adding mailers or changing parsing (i.e., rewriting) or routing information does not require recompilation.

If the mail is being sent by a local user, and the *.mailcf* file exists in the sender's home directory, that file is read as a configuration file after the system configuration file. The primary use of this feature is to add header lines.

The configuration file encodes macro definitions, header definitions, mailer definitions, rewriting rules, and options.

Macros

Macros can be used in three ways. Certain macros transmit unstructured textual information into the mail system, such as the name `sendmail` will use to identify itself in error messages. Other macros transmit information from `sendmail` to the configuration file for use in creating other fields (such as argument vectors to mailers): the name of the sender, and the host and user of the recipient. Other macros are unused internally, and can be used as shorthand in the configuration file.

Header Declarations

Header declarations inform `sendmail` of the format of known header lines. Knowledge of a few header lines is built into `sendmail`, such as the "From:" and "Date:" lines.

Most configured headers will be automatically inserted in the outgoing message if they don't exist in the incoming message. Certain headers are suppressed by some mailers.

Mailer Declarations

Mailer declarations tell `sendmail` of the various mailers available to it. The definition specifies the internal name of the mailer, the pathname of the program to call, some flags associated with the mailer, and an argument vector to be used on the call; this vector is macro-expanded before use.

Address Rewriting Rules

The heart of address parsing in `sendmail` is a set of rewriting rules. These are an ordered list of pattern-replacement rules, (somewhat like a production system, except that order is critical), which are applied to each address. The address is rewritten textually until it is either rewritten into a special canonical form — for example, a (mailer, host, user) 3-tuple, such as {arpanet, usc-isif, postel} representing the address "postel@usc-isif" — or it falls off the end. When a pattern matches, the rule is reapplied until it fails.

The configuration file also supports the editing of addresses into different formats. For example, an address of the form:

```
ucsfcgl!tef
```

might be mapped into:

```
tef@ucsfcgl.UUCP
```

to conform to the domain syntax. Translations can also be done in the other direction.

Option Setting

There are several options that can be set from the configuration file. These include the pathnames of various support files, timeouts, default modes, etc.

E.4. Evaluations and Future Plans

`sendmail` is designed to work in a nonhomogeneous environment. Every attempt is made to avoid imposing unnecessary constraints on the underlying mailers. This goal has driven much of the design. One of the major problems has been the lack of a uniform address space, as postulated in [Postel79a] and [Postel79b].

A nonuniform address space implies that a path will be specified in all addresses, either explicitly (as part of the address) or implicitly (as with implied forwarding to gateways). This restriction has the unpleasant effect of making replying to messages exceedingly difficult, since there is no one "address" for any person, but only a way to get there from wherever you are.

Interfacing to mail programs that were not initially intended to be applied in an internet environment has been amazingly successful, and has reduced the job to a manageable task.

`sendmail` has knowledge of a few difficult environments built in. It generates ARPANET FTP/SMTP compatible error messages (prepended with three-digit numbers [Neigus73, Postel74, Postel82]) as necessary, optionally generates UNIX-style "From" lines on the front of messages for some mailers, and knows how to parse the same lines on input. Also, error handling has an option customized for BerkNet.

The decision to avoid doing any type of delivery where possible (even, or perhaps especially, local delivery) has turned out to be a good idea. Even with local delivery, there are issues of the location of the mailbox, the format of the mailbox, the locking protocol used, etc., that are best decided by other programs. One surprisingly major annoyance in many internet mailers is that the location and format of local mail is built in. The feeling seems to be that local mail is so common that it should be efficient. This feeling is not born out by our experience; on the contrary, the location and format of mailboxes seems to vary widely from system to system.

The ability to automatically generate a response to incoming mail (by forwarding mail to a program) seems useful ("I am on vacation until late August . . .") but can create problems such as forwarding loops (two people on vacation whose programs send notes back and forth, for instance) if these programs are not well written. A program could be written to do standard tasks correctly, but this would solve the general case.

The configuration file is currently practically inscrutable; considerable convenience could be realized with a higher-level format.

In tightly coupled environments, it would be nice to have a name server such as Grapevine [Birrell82] integrated into the mail system. This would allow a site such as "Berkeley" to appear as a single host, rather than as a collection of hosts, and would allow people to move transparently among machines without having to change their addresses. Such a facility would require an automatically updated database and some method of resolving conflicts. Ideally this would be effective even without all hosts being under a single management. However, it is not clear whether this feature should be integrated into the aliasing facility or should be considered a "value added" feature outside `sendmail` itself.

References

[Birrell82] Birrell, A. D., Levin, R., Needham, R. M., and Schroeder, M. D., "Grapevine: An Exercise in Distributed Computing." In *Comm. A.C.M. 25,* 4, April 82.

[Borden79] Borden, S., Gaines, R. S., and Shapiro, N. Z., *The MH Message Handling System: Users' Manual.* R-2367-PAF. Rand Corporation. October 1979.

[Crocker77a] Crocker, D. H., Vittal, J. J., Pogran, K. T., and Henderson, D. A. Jr., *Standard for the Format of ARPA Network Text Messages.* RFC 733, NIC 41952. In [Feinler78]. November 1977.

[Crocker77b] Crocker, D. H., *Framework and Functions of the MS Personal Message System.* R-2134-ARPA, Rand Corporation, Santa Monica, California. 1977.

[Crocker79] Crocker, D. H., Szurkowski, E. S., and Farber, D. J., *An Internetwork Memo Distribution Facility — MMDF.* 6th Data Communication Symposium, Asilomar. November 1979.

[Crocker82] Crocker, D. H., *Standard for the Format of Arpa Internet Text Messages.* RFC 822. Network Information Center, SRI International, Menlo Park, California. August 1982.

[Metcalfe76] Metcalfe, R., and Boggs, D., "Ethernet: Distributed Packet Switching for Local Computer Networks ," *Communications of the ACM 19,* 7. July 1976.

[Feinler78] Feinler, E., and Postel, J. (eds.), *ARPANET Protocol Handbook.* NIC 7104, Network Information Center, SRI International, Menlo Park, California. 1978.

[NBS80] National Bureau of Standards, *Specification of a Draft Message Format Standard.* Report No. ICST/CBOS 80-2. October 1980.

[Neigus73] Neigus, N., *File Transfer Protocol for the ARPA Network.* RFC 542, NIC 17759. In [Feinler78]. August, 1973.

[Nowitz78a] Nowitz, D. A., and Lesk, M. E., *A Dial-Up Network of* UNIX Systems. Bell Laboratories. In UNIX Programmer's Manual,

Seventh Edition, Volume 2. August, 1978.

[Nowitz78b] Nowitz, D. A., *Uucp Implementation Description.* Bell Laboratories. In UNIX Programmer's Manual, Seventh Edition, Volume 2. October, 1978.

[Postel74] Postel, J., and Neigus, N., Revised FTP Reply Codes. RFC 640, NIC 30843. In [Feinler78]. June, 1974.

[Postel77] Postel, J., *Mail Protocol.* NIC 29588. In [Feinler78]. November 1977.

[Postel79a] Postel, J., *Internet Message Protocol.* RFC 753, IEN 85. Network Information Center, SRI International, Menlo Park, California. March 1979.

[Postel79b] Postel, J. B., *An Internetwork Message Structure.* In *Proceedings of the Sixth Data Communications Symposium,* IEEE. New York. November 1979.

[Postel80] Postel, J. B., *A Structured Format for Transmission of Multi-Media Documents.* RFC 767. Network Information Center, SRI International, Menlo Park, California. August 1980.

[Postel82] Postel, J. B., *Simple Mail Transfer Protocol.* RFC821 (obsoleting RFC788). Network Information Center, SRI International, Menlo Park, California. August 1982.

[Schmidt79] Schmidt, E., *An Introduction to the Berkeley Network.* University of California, Berkeley California. 1979.

[Shoens79] Shoens, K., *Mail Reference Manual.* University of California, Berkeley. In UNIX Programmer's Manual, Seventh Edition, Volume 2C. December 1979.

[Sluizer81] Sluizer, S., and Postel, J. B., *Mail Transfer Protocol.* RFC 780. Network Information Center, SRI International, Menlo Park, California. May 1981.

[Solomon81] Solomon, M., Landweber, L., and Neuhengen, D., "The Design of the CSNET Name Server." CS-DN-2, University of Wisconsin, Madison. November 1981.

[Su82] Su, Zaw-Sing, and Postel, Jon, *The Domain Naming Convention for Internet User Applications.* RFC819. Network Information Center, SRI International, Menlo Park, California. August 1982.

[UNIX83] *The* UNIX Programmer's Manual, Seventh Edition, Virtual VAX-11 Version, Volume 1. Bell Laboratories, modified by the University of California, Berkeley, California. March, 1983.

Index

Special Characters

`.rhosts`, 110
`/etc/nd.local`, 10 *thru* 13
`/etc/nd.local` commands
- `clear`, 12
- `serverat`, 12
- `soff`, 12
- `son`, 12
- `user`, 11
- `version`, 12

1

1/2 inch tapes, 81
1/4 inch tapes, 81

A

abort printer, 169
accounting, 212
adding
- clients to `nd` server, 59 *thru* 65
- new users, 56 *thru* 58

adding boards, 138 *thru* 141
- configuring kernel, 138
- making entries in /dev, 139

adding hardware, 137 *thru* 173
adding modem, 149 *thru* 157
- Hayes Smartmodem, 150
- Ven-Tel modem, 149

adding printer, 158 *thru* 173
- editing `printcap`, 159
- remote spooler, 166
- VPC-2200 Multibus Board, 162

adding terminal, 145 *thru* 148
administering printer, 168 *thru* 173
- abort printer, 169
- disable printer, 169
- enable printer, 169
- `lpc` — printer control program, 169
- `lpd` — printer daemon, 168
- restart printer, 169
- start printer, 169
- stop printer, 169
- topq — move jobs to top of queue, 169

administering `uucp`, 323 *thru* 325
- file modes, 325
- LCK, 324
- LOG, 323

administering `uucp`, *continued*
- login entry, 325
- shell files, 324
- SQFILE, 323
- STST, 324
- TM, 323

alias (yellow pages) for `sendmail`, 336
alias database for `sendmail`, 335
alternate configuration file for `sendmail`, 338
arguments to `sendmail`, 337 *thru* 338
- alternate configuration file, 338
- changing options, 338
- daemon mode, 337
- debugging, 338
- force queue, 338
- queue interval, 337

Arpanet forwarding for mail, 127
asynchronous serial ports
- connecting devices to, 142 *thru* 144

B

backing up `ndl` partitions, 82
backing up file systems, 81 *thru* 86, 178 *thru* 179
backing up file systems over Ethernet, 86
booting `diag`, 227
bootstrap and shutdown, 180 *thru* 196

C

call remote system in `uucico`, 312
call unit info for `uucp`, 317
cartridge tapes, 81
changing options for `sendmail`, 338
check quotas — `quotacheck`, 95
checking connectivity in file system, 265
checking free blocks in file system, 263
checking inode data in file system, 265
checking inode data size in file system, 264
checking inode links in file system, 264
checking inode states in file system, 263
checking super-block in file system, 263
`clear diag` command, 247
client, 8
clients
- adding to `nd` server, 59 *thru* 65

command line flags for `sendmail`, 355 *thru* 356
command line for `uux`, 310

commands in /etc/nd.local
clear, 12
serverat, 12
soff, 12
son, 12
user, 11
version, 12
communications, 101 *thru* 133
Arpanet forwarding for mail, 127
configuring for mail forwarding, 126
domains for mail routing, 124
Ethernet, 102 *thru* 116
mail routing, 123
main machine for mail routing, 125
postmaster alias for mail, 127
testing mail configuration, 129
tip command, 131
troubleshooting mail system, 129
uucp, 117
wide area networks, 117 *thru* 122
yellow pages for mail, 128
compatibility
in network, 112
conditionals for sendmail, 344
configuration file (alternate) for sendmail, 338
configuration file for sendmail, 341 *thru* 354
conditionals, 344
define classes, 342
define header, 343
define macro, 342
define mailer, 343
define trusted users, 344
left hand side, 346
precedences, 344
rewriting rules, 342
right hand side, 347
set option, 343
special classes, 346
special macros, 344
syntax, 342
configuration options for sendmail, 357 *thru* 358
configuring diag, 229
configuring for mail forwarding, 126
configuring kernel, 201 *thru* 208
configuring kernel to add new boards, 138
connecting
devices to serial ports, 142 *thru* 144
connectivity in file system
checking, 265
controller for a disk, 70
controlling resources, 211
conversation termination in uucico, 314
corrupted file system, 262
crashes, 197 *thru* 200
creating
home directory, 58
cylinder groups, 260
cylinders on a disk, 69

D

daemon mode for sendmail, 337
debugging for sendmail, 338
define classes for sendmail, 342
define header for sendmail, 343
define macro for sendmail, 342
define mailer for sendmail, 343
define trusted users for sendmail, 344
delivery mode for sendmail, 339
devices types for uucp, 322
diag
format command, 234
label command, 236, 239
diag commands
clear, 247
diag, 247
dmatest, 247
fix, 249
format, 248
help, 247
label, 252
map, 249
partition, 253
position, 247
quit, 247
read, 247
rhdr, 250
scan, 255
seek, 247
slip, 250
status, 247
test, 247
translate, 247
verify, 247
version, 247
whdr, 256
write, 247
diag diag command, 247
diag — disk maintenance, 225 *thru* 256
booting, 227
configuring, 229
preparing new disks, 233
dialcode abbreviations file for uucp, 318
disable printer, 169
disable quotas — quotaoff, 95
disk maintenance
booting diag, 227
configuring diag, 229
diag program, 225, 256
preparing new disks, 233
Disk Partitioning
Default Sizing, 240
disk partitions, 10
disk quotas for file systems, 94 *thru* 97
edquota — set quotas, 94
quotacheck — check quotas, 95
quotaoff — turn off quotas, 95
quotaon — turn on quotas, 94
repquota — report quotas, 95
disks, 69 *thru* 98
Disks

Disks, *continued*
168 MByte Disk Partitions, 240
169 MByte Disk Partitions, 240
474 MByte Disk Partitions, 240
50 MByte Disk Partitions, 237
84 MByte Disk Partitions, 240
86 MByte Disk Partitions, 237
disks
controller, 70
cylinders, 69
formatting SCSI devices, 72
formatting SMD devices, 71
heads, 69
labels, 73
mapping SMD devices, 72
partitions, 73
platters, 69
SCSI devices, 71
sectors, 70
slipping SCSI devices, 72
SMD devices, 70
tracks, 69
units, 70
`dmatest diag` command, 247
domains for mail routing, 124
`dump` — backing up file systems, 81 *thru* 86

E

editing `printcap` for new printers, 159
`edquota` — set disk quotas, 94
enable printer, 169
enable quotas — `quotaon`, 94
errors from `fsck`, 265 *thru* 279
`/etc/hosts.equiv`, 109
`/etc/nd.local`, 10 *thru* 13
`/etc/nd.local` commands
`clear`, 12
`serverat`, 12
`soff`, 12
`son`, 12
`user`, 11
`version`, 12
`/etc/remote`, 131
Ethernet, 102 *thru* 116
backing up file systems over, 86
network security, 109
reducing overhead, 107
restoring file systems over, 91
setting up gateway, 104

F

file modes for `sendmail`, 341
file modes for `uucp`, 325
file system
checking connectivity, 265
checking free blocks, 263
checking inode data, 265
checking inode data size, 264
checking inode links, 264
checking inode states, 263
checking super-block, 263
corruption, 262
file system, *continued*
cylinder groups, 260
fragments, 261
summary information, 260
super-block, 259
updating, 261
file system check program, 259 *thru* 280
errors, 265 *thru* 279
file systems, 69 *thru* 98
backing up, 81 *thru* 86
backing up over Ethernet, 86
`edquota` — set quotas, 94
maintaining, 93 *thru* 98
`quotacheck` — check quotas, 95
`quotaoff` — turn off quotas, 95
`quotaon` — turn on quotas, 94
quotas, 94 *thru* 97
`repquota` — report quotas, 95
restoring, 87 *thru* 93
restoring damaged root, 89
restoring entire, 88
restoring over Ethernet, 91
restoring specific files, 87
files
restoring specific, 87
files needing periodic attention, 214
files required for `uucp`, 317
L-devices, 317
L-dialcodes, 318
L.cmds, 322
L.sys, 320
USERFILE, 318
files to support `sendmail`, 361
filters for printer spooler, 167
`fix diag` command, 249
Fix SMD Disks, 237 *thru* 238
flags for `sendmail` command line, 355 *thru* 356
force queue for `sendmail`, 338
`format diag` command, 248
formatting
SCSI devices, 72
SMD devices, 71
Formatting SCSI Disks
PAGE, 234
Formatting Subcommands, SCSI Disks, 234
`.forward` files for `sendmail`, 337
fragments
in file system, 261
free blocks in file system
checking, 263
`fsck` error conditions, 265 *thru* 279
`fsck` — file system check program, 259 *thru* 280
Fujitsu 2284 SMD 14-inch Disk
Partition Sizes, 240
Fujitsu 2312K SMD 8-inch Disk
Partition Sizes, 240
Fujitsu 2322 SMD 8-inch Disk
Partition Sizes, 240
Fujitsu 2351 SMD Eagle Disk
Partition Sizes, 240
Fujitsu M2243AS SCSI 5-1/4-inch Disk

Fujitsu M2243AS SCSI 5-1/4-inch Disk, *continued*
Partition Sizes, 237

G

gateway
setting up, 104

H

hard partitions, 10
Hayes Smartmodem
adding to system, 150
support for `tip`, 155
header lines for `sendmail`, 337
heads on a disk, 69
`help diag` command, 247
home directory
creating, 58

I

inode data in file system
checking, 265
inode data size in file system
checking, 264
inode links in file system
checking, 264
inode states in file system
checking, 263
installing `sendmail`, 329 *thru* 361
alternate configuration file, 338
arguments, 337 *thru* 338
basic installation, 329 *thru* 331
changing options, 338
command line flags, 355 *thru* 356
configuration file, 341 *thru* 354
configuration options, 357 *thru* 358
daemon mode, 337
debugging, 338
force queue, 338
mailer flags, 359 *thru* 360
normal operations, 332 *thru* 337
queue interval, 337
support files, 361
tuning, 338 *thru* 341
installing `uucp`, 119, 315 *thru* 323
compiling system, 317
modifying `Makefile`, 316
modifying `uucp.h`, 316
required files, 317
internetwork mail routing, 365 *thru* 376

K

kernel
configuring, 201, 208
configuring to add new boards, 138

L

`L-devices` file for `uucp`, 317
`L-dialcodes` file for `uucp`, 318
`L.cmds` file for `uucp`, 322
`L.sys` file for `uucp`, 320
`label diag` command, 252
Labeling SCSI Disks, 236 *thru* 237
Labeling SMD Disks, 239 *thru* 240
labels on disks, 73
left hand side for `sendmail`, 346
level of log for `sendmail`, 340
limiting load for `sendmail`, 340
line printer administration, 168 *thru* 173
abort printer, 169
disable printer, 169
enable printer, 169
`lpc` — printer control program, 169
`lpd` — printer daemon, 168
restart printer, 169
start printer, 169
stop printer, 169
topq — move jobs to top of queue, 169
line protocol selection in `uucico`, 312
load limiting for `sendmail`, 340
local alias database for `sendmail`, 335
local modifications, 213
lock files for `uucp`, 324
log entry files for `uucp`, 323
log level for `sendmail`, 340
log, system, for `sendmail`, 332
login entry for `uucp`, 325
`lpc` — printer control program, 169
`lpd` — printer daemon, 168
`lpq` — show print queue, 170
`lpr` — spool print jobs, 170
`lprm` — remove print jobs, 170

M

mail
testing configuration, 129
troubleshooting, 129
mail forwarding
configuring for, 126
in Arpanet, 127
postmaster alias, 127
yellow pages, 128
mail queue for `sendmail`, 332
mail routing, 123, 365 *thru* 376
mail routing domains, 124
mail routing main machine, 125
mailer flags for `sendmail`, 359 *thru* 360
main machine for mail routing, 125
maintaining file systems, 93 *thru* 98
maintenance, 177 *thru* 214
accounting, 212
backing up file systems, 178 *thru* 179
bootstrap and shutdown, 180 *thru* 196
configuring kernel, 201 *thru* 208
controlling resources, 211
crashes, 197 *thru* 200
files needing attention, 214
local modifications, 213
monitoring performance, 210
system logs, 209
MAKEDEV command, 139
`map diag` command, 249

mapping
SMD, 227
SMD devices, 72
Maxtor XT-1050 SCSI 5-1/4-inch Disk
Partition Sizes, 237
merging old files when upgrading, 220
Micropolis 1304 SCSI 5-1/4-inch Disk
Partition Sizes, 237
Micropolis 1325 SCSI 5-1/4-inch Disk
Partition Sizes, 237
modem
adding Hayes Smartmodem to system, 150
adding to system, 149, 157
adding Ven-Tel modem to system, 149
modes of files for `sendmail`, 341
modes on files for `uucp`, 325
monitoring performance, 210

N

`nd` — network disk service, 10 *thru* 17
`ndl` partitions
backing up, 82
`nd` server
adding clients to, 59 *thru* 65
network compatibility, 112
network disk service nd, 10 *thru* 17
network file system service — NFS, 17 *thru* 35
network security, 109
`.rhosts`, 110
`/etc/hosts.equiv`, 109
network services, 7 *thru* 65
`nd`, 10 *thru* 17
NFS — network file system, 17 *thru* 35
`yp` — yellow pages, 35 *thru* 56
networks
Ethernet, 102 *thru* 116
reducing overhead, 107
setting up gateway, 104
`uucp`, 117
wide area, 117, 122
new users
adding to machine, 56 *thru* 58
NFS — network file system service, 17 *thru* 35
normal operations of `sendmail`, 332 *thru* 337
null modem, 142

O

operating `sendmail`, 329 *thru* 361
options for configuring `sendmail`, 357 *thru* 358
output filters for printer, 167

P

`partition diag` command, 253
Partition Sizes, 240
partitions, 10
partitions on disks, 73
per-user forwarding for `sendmail`, 337
performance monitoring, 210
periodic maintenance, 177 *thru* 214
accounting, 212
periodic maintenance, *continued*
backing up file systems, 178 *thru* 179
bootstrap and shutdown, 180 *thru* 196
configuring kernel, 201 *thru* 208
controlling resources, 211
crashes, 197 *thru* 200
files needing attention, 214
local modifications, 213
monitoring performance, 210
system logs, 209
platters on a disk, 69
`position diag` command, 247
postmaster alias for mail, 127
precedences for `sendmail`, 344
preparing new disks, 233
`printcap`
editing for new printers, 159
printer
adding to system, 158, 173
editing `printcap`, 159
hooking up to VPC-2200 Multibus Board, 162
`lpq` — show print queue, 170
`lpr` — spool print jobs, 170
`lprm` — remove print jobs, 170
output filters, 167
remote spooler, 166
printer administration, 168 *thru* 173
abort printer, 169
disable printer, 169
enable printer, 169
`lpc` — printer control program, 169
`lpd` — printer daemon, 168
restart printer, 169
start printer, 169
stop printer, 169
topq — move jobs to top of queue, 169
process work in `uucico`, 313

Q

queue forcing for `sendmail`, 338
queue interval for, 337
queue, mail, for `sendmail`, 332
`quit diag` command, 247
`quotacheck` — check disk quotas, 95
`quotaoff` — turn off disk quotas, 95
`quotaon` — turn on disk quotas, 94
quotas for file systems, 94 *thru* 97
`edquota` — set quotas, 94

R

`read diag` command, 247
reducing network overhead, 107
reel to reel tapes, 81
remote spooler, 166
remove print jobs — `lprm`, 170
report quotas — `repquota`, 95
`repquota` — report disk quotas, 95
required file line for `uux`, 309
resources
controlling, 211
restart printer, 169

`restore` — restoring file systems, 87 *thru* 93
restoring damaged root file systems, 89
restoring entire file systems, 88
restoring file systems, 87 *thru* 93
restoring file systems over Ethernet, 91
restoring specific files, 87
rewriting rules for `sendmail`, 342
`rhdr diag` command, 250
right hand side for `sendmail`, 347

S

`scan diag` command, 255
scan for work in `uucico`, 311
SCSI devices
    formatting, 72
    slipping, 72
SCSI disk, 71
SCSI Disks
    Defect List, 235
    Formatting, 234 *thru* 236
    Formatting Subcommands, 234
    Labeling, 236 *thru* 237
    Surface Analysis, 235
sector slipping, 227
sectors on a disk, 70
security
    in network, 109
security in network
    `.rhosts`, 110
    `/etc/hosts.equiv`, 109
security in `uucp`, 315
`seek diag` command, 247
select line protocol in `uucico`, 312
`sendmail`, 365 *thru* 376
    `.forward` files, 337
    installation and operation, 329 *thru* 361
    local alias database, 335
    mail queue, 332
    per-user forwarding, 337
    special header lines, 337
    system log, 332
    yellow pages aliases, 336
`sendmail` arguments
    alternate configuration file, 338
    changing options, 338
    daemon mode, 337
    debugging, 338
    force queue, 338
    queue interval, 337
`sendmail` configuration file
    conditionals, 344
    define classes, 342
    define header, 343
    define macro, 342
    define mailer, 343
    define trusted users, 344
    left hand side, 346
    precedences, 344
    rewriting rules, 342
    right hand side, 347
    set option, 343
`sendmail` configuration file, *continued*
    special classes, 346
    special macros, 344
    syntax, 342
`sendmail` installation
    arguments, 337 *thru* 338
    basic installation, 329 *thru* 331
    command line flags, 355 *thru* 356
    configuration file, 341 *thru* 354
    configuration options, 357 *thru* 358
    mailer flags, 359 *thru* 360
    normal operations, 332 *thru* 337
    support files, 361
    tuning, 338 *thru* 341
`sendmail` tuning
    delivery mode, 339
    file modes, 341
    load limiting, 340
    log level, 340
    timeouts, 339
sequence check file for `uucp`, 323
serial ports
    connecting devices to, 142 *thru* 144
server, 8
    adding clients to `nd`, 59 *thru* 65
set option for `sendmail`, 343
setting up `diag`, 229
setting up gateway, 104
setting up mail routing, 123
setting up mail routing domains, 124
shell files for `uucp`, 324
show print queue — `lpq`, 170
`slip diag` command, 250
slipping
    SCSI devices, 72
SMD devices
    formatting, 71
    mapping, 72
SMD disk, 70
SMD Disks
    Labeling, 239 *thru* 240
    Surface Analysis, 237 *thru* 239
soft partitions, 10
special classes for `sendmail`, 346
special header lines for `sendmail`, 337
special macros for `sendmail`, 344
spool print jobs — `lpr`, 170
spooler
    remote, 166
standard input line for `uux`, 310
standard output line for `uux`, 310
start printer, 169
`status diag` command, 247
stop printer, 169
summary information
    for file system, 260
super-block, 259
super-block in file system
    checking, 263
support files for `sendmail`, 361
Surface Analysis, 235

syntax of configuration file for `sendmail`, 342
system log for `sendmail`, 332
system logs, 209
system maintenance, 177 *thru* 214
    accounting, 212
    backing up file systems, 178 *thru* 179
    bootstrap and shutdown, 180 *thru* 196
    configuring kernel, 201 *thru* 208
    controlling resources, 211
    crashes, 197 *thru* 200
    files needing attention, 214
    local modifications, 213
    monitoring performance, 210
    system logs, 209
system status files for `uucp`, 324

T

tapes
    cartridge, 81
    reel to reel, 81
temporary data files for `uucp`, 323
terminal
    adding to system, 145, 148
terminate conversation in `uucico`, 314
`test diag` command, 247
testing mail configuration, 129
`tip` command, 131
    `/etc/remote`, 131
tip with Hayes Smartmodem, 155
topq — move jobs to top of printer queue, 169
tracks on a disk, 69
`translate diag` command, 247
troubleshooting mail system, 129
tuning `sendmail`, 338 *thru* 341
    delivery mode, 339
    file modes, 341
    load limiting, 340
    log level, 340
    timeouts, 339

U

units on a disk controller, 70
UNIX file system, 283 *thru* 301
updating file system, 261
upgrading system software, 217 *thru* 221
    files to save, 218
    merging old files, 220
user line for `uux`, 309
`USERFILE` file for `uucp`, 318
users
    adding new to machine, 56 *thru* 58
`uucico`
    call remote system, 312
    process work, 313
    scan for work, 311
    select line protocol, 312
    terminate conversation, 314
`uucico` — UNIX copy in copy out, 310 *thru* 314
`uuclean` — clean spool area, 314 *thru* 315
`uucp` — UNIX to UNIX copy, 117
    commands, 117
`uucp` — UNIX to UNIX copy, *continued*
    files and commands, 118
    installing, 119
    security, 315
    spooling area, 118
uucp administration, 323 *thru* 325
    file modes, 325
    `LCK`, 324
    `LOG`, 323
    login entry, 325
    shell files, 324
    `SQFILE`, 323
    `STST`, 324
    `TM`, 323
uucp device types, 322
uucp files
    L-devices, 317
    L-dialcodes, 318
    L.cmds, 322
    L.sys, 320
    USERFILE, 318
uucp implementation, 305 *thru* 325
uucp installation, 315 *thru* 323
    compiling system, 317
    modifying `Makefile`, 316
    modifying `uucp.h`, 316
    required files, 317
uucp — UNIX file copy, 305 *thru* 308
`uulog` — `uucp` log inquiry, 314
`uux`
    command line, 310
    required file line, 309
    standard input line, 310
    standard output line, 310
    user line, 309
`uux` — UNIX to UNIX execute, 308 *thru* 310
`uuxqt` — UNIX command execute, 310

V

Ven-Tel modem
    adding to system, 149
`verify diag` command, 247
`version diag` command, 247
VPC-2200 Multibus Board
    hooking up for printer, 162

W

`whdr diag` command, 256
wide area networks, 117 *thru* 122
    uucp, 117
work processing in `uucico`, 313
`write diag` command, 247

Y

yellow pages aliases for `sendmail`, 336
yellow pages domain, 36
yellow pages for mail, 128
yellow pages map, 36
yellow pages master, 36
yellow pages service — `yp`, 35 *thru* 56
yellow pages slave, 36

yp — yellow pages service, 35 *thru* 56

Revision History

Revision	Date	Comments
1	19 November 1984	First α release of this System Administration Manual.
2	14 February 1985	Release α.1 incorporating review comments, but not reflecting effects of NFS.
3	16 March 1985	Release α.2 incorporating NFS and *yp* information.
50	29 March 1985	First β release of this System Administration Manual.
A	15 April 1985	First customer release of this System Administration Manual.
4	15 August 1985	Second α release of this System Administration Manual, for 3.0.
51	15 October 1985	Second β release of this System Administration Manual, for 3.0.
B	15 February 1986	Second customer release of this System Administration Manual, for 3.0.

Notes

Notes

Notes